MELTS: A framework for applying machine learning to time series
MELTS: Un marco de trabajo para la aplicación de aprendizaje automático a series temporales

Trabajo de Fin de Máster
Curso 2019–2020

Autor
Simona Florina Puica

Director
Rafael Caballero Roldán

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

MELTS: A framework for applying machine learning to time series
MELTS: Un marco de trabajo para la aplicación de aprendizaje automático a series temporales

Trabajo de Fin de Máster en Ingeniería Informática
Departamento de Sistemas Informáticos y Computación

Autor
Simona Florina Puica

Director
Rafael Caballero Roldán

Convocatoria: *Junio-Julio 2020*
Calificación: *10 - Matrícula de honor*

Máster en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

2 de julio de 2020

Resumen

MELTS: Un marco de trabajo para la aplicación de aprendizaje automático a series temporales

El análisis de series temporales es un área de investigación de mucha relevancia desde hace casi un siglo, con muchas técnicas desarrolladas para predecir y analizar datos temporales obtenidos de diversos campos como por ejemplo el de la economía, sociología o biología. Más recientemente, en el área conocida como aprendizaje automático se han ido proponiendo un conjunto de técnicas muy generales dedicadas al análisis y predicción de datos en un contexto genérico, no temporal. Aunque ha habido muchos esfuerzos para aplicar técnicas de aprendizaje automático a datos de series temporales, las conexiones temporales presentes entre los datos dan lugar a ciertas sutilezas que introducen dificultades en las fases de entrenamiento y test tradicionales empleadas por las técnicas estándar de aprendizaje automático, y dichas sutilezas se tienen que tener en cuenta.

En este documento se propone un marco de trabajo para realizar análisis de series temporales mediante el uso de algoritmos de aprendizaje automático. En particular, este trabajo se ha enfocado en la detección de valores atípicos (outliers) en series temporales multivariable.

El marco de trabajo propuesto contiene una técnica de preprocesado que permite preparar los datos para el empleo de cualquier método de aprendizaje automático supervisado de clasificación. Esta fase combina varias filas de atributos en una sola, que además incluye el valor futuro a predecir. Esto permite que los algoritmos de aprendizaje automático infiera la relación entre varios atributos del pasado y un valor futuro. También se presenta una propuesta para el entrenamiento y test de modelos con estos datos preprocesados con el fin de evaluar el desempeño del modelo.

Un conjunto de datos pertenecientes al mercado de valores se ha empleado como caso de uso. Adicionalmente, en este documento también se presentan unas adaptaciones de método específicas, se definen métricas a medida para evaluar la ganancia y se realiza un análisis sobre el impacto que los hiperparámetros tienen sobre la eficacia del método elegido.

Palabras clave

aprendizaje automático, series temporales, preprocesado, entrenamiento y test, detección de valores atípicos, detección de outliers, mercado de valores

Abstract

MELTS: A framework for applying machine learning to time series

Time series analysis has been a relevant field of research for close to a century, with many different techniques developed for predicting and analysing the temporal data obtained in areas such as economics, sociology, biology, etc. More recently, machine learning has been proposed as a set of very general techniques devoted to analysing and predicting data in a more general, non-temporal, context. Although there have been many efforts for applying machine learning techniques to time series data, the temporal connections among the data raise certain subtleties that introduce difficulties in the traditional training and test phases employed in standard machine learning, and must be taken into account.

This work proposes a framework for analysing time series within machine learning. In particular, the framework is focused on outlier detection in multivariate time series.

The proposed framework contains a preprocessing technique for preparing the data for employing any machine learning supervised classification method. This phase combines several rows of features into a single one that includes one future value to predict. This allows the machine learning method to infer the relationship among several past features and one future value. We also present a proposal for training and testing models with this preprocessed data in order to check how well the model performs.

A stock market dataset is used as a support use case. Additionally, we also present method-specific adaptations, define custom metrics for evaluating the gain, and perform an analysis on how hyperparameters impact the efficacy of the method.

Keywords

machine learning, ML, time series, preprocessing, train and test, outlier detection, stock market

Contents

List of figures

List of tables

Chapter 1

Introduction

In this chapter we present the motivations of this work and explore the state of the art to get an overview of the topic. Afterwards, we establish the objectives of the project and present the methodology that has been followed in order to reach them.

1.1. Motivation

Machine learning (in short ML, Bishop (2006)) has increased in popularity in the past few years. It should not be a surprise, since it has so many applications, that there are many courses and tutorials available for beginners or experts alike, online and offline, paid or free of charge, teaching different aspects and approaches to ML.

However, not many of them show how to apply these techniques to time series. In particular, the data preparation process and the evaluation of results are well established for general machine learning, but in the case of time series, the data has a time dependency component that can require extra effort to ensure a correct application and sensible results, and these details are often disregarded.

This work aims to explore how would the data need to be prepared, and how should the results be evaluated, if a ML technique were to be applied to a time series dataset.

Although we present a general framework for preprocessing and evaluating time series datasets in the context of ML, a stock market dataset is employed as a support use case in order to check that the results found during this research work are promising.

1.2. State of the art

1.2.1. Time series and outliers

A time series is a series of data points that are ordered in time. A few examples of time series are data collected by sensors in internet of things (IoT) systems, the hourly heart rate of a patient, monthly rainfall for a specific area, or the daily number of passengers on a train. Depending on the amount of variables, time series can be univariate (only one variable) or multivariate (multiple time-dependent variables).

Time series analysis has been an important topic for almost one century, and it predates modern ML techniques. Because of this, there are many well-established and widely accepted techniques available for time series analysis and forecasting, such as ARMA (Makridakis and Hibon (1997)) or ARCH (Bollerslev et al. (1994)) model families.

By analyzing time series data, we expect to extract non-obvious information out of it, and use such information when processing new incoming data. This can improve tasks such as forecasting, classification or outlier detection, among others.

Out of the aforementioned tasks, the last one, outlier detection, will be the focus of this project. In statistics, an outlier is defined as a value of the dataset that deviates from the other values by a large margin. This definition, as well as the following outlier classification, has been extracted from Blázquez-García et al. (2020).

In the context of time series, outliers can be classified as *point outliers*, *subsequence outliers* or *outlier time series*. Depending on the type, different methods have been proposed in the literature for detecting outliers in time series data. Below we have an overview of some of the proposed methods for each type:

1. *Point outliers*: A specific point behaves in an atypical manner in comparison to its neighbours, in which case it will be a *local outlier*, or in comparison to the whole time series, which will then be a *global outlier*. The detection methods for point outliers can be classified as follows:

    - In univariate series:
        - Model-based methods: estimation models (Basu and Meckesheimer (2007) used the median, while Mehrang et al. (2015) used the Mean Absolute Deviation (MAD)) or prediction models (Reddy et al. (2017) used Gaussian Mixture Models (GMM), and Carter and Streilein (2012) used Exponentially Weighted Moving Average (EWMA)).
        - Density-based methods: In this case, an outlier is defined as a point with fewer than a given amount of neighbours. However, the concept of *neighbour* in time series is complex and not easy to define. Angiulli and Fassetti (2007) and Ishimtsev et al. (2017) took the temporality of the data into account when developing their density-based methods.
        - Histogramming: Jagadish et al. (1999) and Muthukrishnan et al. (2004) proposed different approaches on detecting the points that, if removed from the univariate time series, would lower the error in the series' histrogam representation.
    - In multivariate time series, the outlier detection can be done on one or multiple variables together:
        - *Univariate outlier detection in multivariate time series*:
            - Since a multivariate series can be considered an aggregation of multiple time-dependent variables, the same techniques proposed in the above points can also be applied for detecting outliers in individual variables.
            - However, when considering the different variables independently, a loss of information can happen. To overcome this issue, different techniques based on reducing the dimensionality of the multivariate time series have been proposed; after reducing the dimensions, applying the standard detection techniques for univariate series would be possible. Some of these techniques look for a new set of uncorrelated variables, such as those proposed by Papadimitriou et al. (2005) or Baragona and Battaglia (2007).
        - *Multivariate outlier detection*: In this case the methods deal with multiple variables at the same time. Different approaches have been proposed:

- Model-based methods: Su et al. (2019) and Sakurada and Yairi (2014) used estimation models based on autoencoders, while others used prediction models such as Contextual Hidden Markov Model (CHMM) (Zhang et al. (2016)) or Convolutional Neural Networks (CNN) (Munir et al. (2018)).
- Dissimilarity-based models: Cheng et al. (2008) and Li et al. (2009).
- Histogramming: Muthukrishnan et al. (2004) extended the method based on histogramming mentioned earlier to also include multivariate time series.

2. *Subsequence outliers*: Multiple consecutive points in the time series seem unusual as a group, but each one seems normal enough to not be considered an outlier value when observed isolated. There are different proposed outlier detection methods:

   - For univariate time series, proposed methods are based on *discord detection* (Keogh et al. (2005), Lin et al. (2005)), *dissimilarity* (Chen and Cook (2011), Izakian and Pedrycz (2013)), *prediction models* (Munir et al. (2018)), *frequency* (Rasheed and Alhajj (2013)) and *information theory* (Yang et al. (2001) or Yang et al. (2004)).
   - For multivariate series, the same distinction as for the point outliers applies:
     - *Univariate outlier detection*: Similarly to the aforementioned case for point outliers, the techniques used to detect subsequence outliers in univariate time series can be applied to this case as well. Additionally, approaches that consider dimension reduction, such as Wang et al. (2018) or Hu et al. (2019), have also been proposed for this case.
     - *Multivariate outlier detection*: Different methods based on *estimation models* (Jones et al. (2014)), *prediction models* (Munir et al. (2018)) and *dissimilarity* (Cheng et al. (2008)).

3. *Outlier time series*: in the case of a multivariate time series, it can happen that one of the variables, which in turn is also a time series, is an outlier when compared to the others. Fewer methods have been proposed for detecting this type of outliers than for the previous ones. Out of the proposals, some are based on *dimensionality reduction* (Laptev et al. (2015) and Hyndman et al. (2015)), and others are based on *dissimilarity*, with clustering being the most common technique (Rebbapragada et al. (2009), Benkabou et al. (2018)).

1.2.2. Prediction in economy

Among many other fields, time series analysis has been of special interest for economy. In particular, there has been a growing interest in using ML techniques for stock market prediction.

For example, Nelson et al. (2017) studied the usage of Long Short-Term Memory (LSTM) neural networks (Hochreiter and Schmidhuber (1997)) to predict future trends of stock prices based on the price history, alongside with technical analysis indicators and obtained up to an average of 55.9% of accuracy when predicting if the price of a particular stock is going to go up or not in the near future. Akita et al. (2016) also used LSTM, but they went one step further and added textual information from news articles related to the companies whose stock data they were using.

In this project, we use stock market data not with the intention of improving the already existing techniques, but with the purpose of testing whether the proposed framework is applicable to a practical use case.

1.3. Objectives

As the main title says, the end goal of this project is to develop a framework for applying ML methods to time series. In this section we break down this main goal into more concrete, approachable objectives:

- **Define a general preprocessing framework for using classic ML methods applied to time series.** As opposed to other datasets, time series data has a time dependency that needs to be maintained during the data preprocessing stage and taken into consideration when processing the raw input data. If we do not handle the data properly, we risk losing the time relationship among the data and getting inaccurate results as a side effect.

- **Define an evaluation method for checking the goodness of the selected ML method.** Since the framework is designed so that *any* ML algorithm can be applied to the preprocessed files, it is important to offer the user a way of checking whether the selected method is giving good results or not.

- **Apply the previous two points to a use case**, to see whether our approach is viable and whether we get coherent results in the following aspects:

  - **Viability**:
    - We obtain positive results (i.e. we gain and not lose). If this happens, it means that it is a path worth exploring (by using other methods, hyperparameters, etc.) in order to reach a significant gain.
  - **Monotony**:
    - Backward: The system has better results when using more past data.
    - Forward: The system has worse results when predicting values further away into the future.

1.3.1. Scope limitation

Due to time constrains, the scope of this project was limited in the following aspects:

1. **Supervised learning**

   Machine learning is a very broad topic, so in this project we focused on working with supervised learning algorithms (Ayodele (2010)) as a more specific area of ML. Supervised learning aims at predicting a specific column of the dataset, commonly called the *label*. The system first *trains* a model with data that includes the label, thus inferring the relations between the rest of the columns, usually called the *features* and the label. Then, the model is used to predict the label for new data which only includes these features. The individual values contained in the label are referred to as *class*, commonly represented with consecutive numbers (0, 1, 2, etc.).

2. **Outlier detection**

   The dataset for our use case (further explained in Section 2.2) corresponds a multivariate time series. In this case we want to predict univariate global outliers in a multivariate time series, and we will use model-based outlier detection techniques. Thus, we have a binary label with the following two classes: 0 representing that the row does not correspond to an outlier and 1 otherwise. This label is not directly in our row data and will be generated.

1.4. Methodology and document structure

In the previous section we established the objectives of this project. In order to reach those goals, the Cross-Industry Standard Process for Data Mining (CRISP-DM, Wirth and Hipp (2000)) methodology has been chosen. Besides being robust and well-proven, CRISP-DM is also widely used in data science projects, which makes it suitable for this project. This process is divided into six major steps that define an idealised path (see Figure 1.1[1]), while allowing flexibility when applying it to a specific project or scenario.

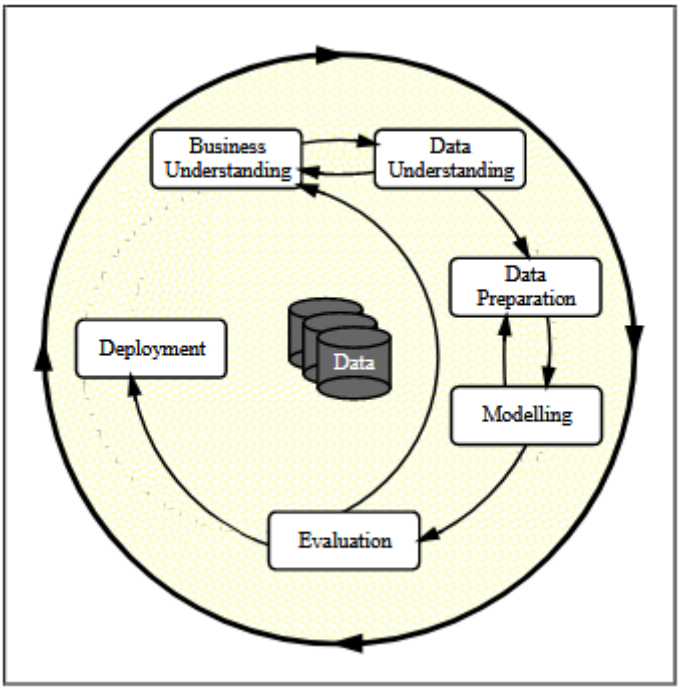


Figure 1.1: CRISP-DM process diagram

In the list below there is a brief overview of the steps involved and how they were applied to this project:

1. **Business understanding**

   The process starts with approaching the project from a business perspective. This involves understanding the purpose of the analysis in order to define the scope of the project, and defining the business metrics by which the results will be evaluated.

[1]Source: Wirth and Hipp (2000)

The outcome of this step can be read in Chapter 2, which describes in depth the business case that was used in this project and introduces two new metrics suitable for this case.

2. **Data understanding**

   This step involves first defining the data sources and gathering the required data, and then understanding it and discovering preliminary insights into the data.

   In the case of this project the data had already been provided from the beginning and the data understanding was covered also in Chapter 2, in Section 2.2.

3. **Data preparation**

   This step covers all the data cleanup and preparation, since the raw data needs to be cleaned and preprocessed before it is ready in order to get a dataset ready for further analysis.

   This was one of the core steps in this project, as time series data requires a different processing for ML. Chapter 3 is fully dedicated to data preprocessing.

4. **Modeling**

   This step usually involves feature engineering, selecting modeling techniques and possibly creating a proof-of-concept model. Since the focus of this project was to design a framework rather than doing actual data analysis, this step has been simplified to picking a few ML methods for testing purposes.

5. **Evaluation**

   This step is meant to check the quality or validity of the model previously designed.

   Chapter 4 covers this step from a theoretical point of view, and the results of the application to our specific business case can be found in Chapter 5. This is another key contribution of this work, since the evaluation of time series with ML needs specific techniques not covered in the literature to the best of our knowledge.

6. **Deployment**

   As a conclusion of the project, ideally there would be a full deployment of the resulting system as an application that could operate over new incoming data.

   Since this is not part of the goals of this project, this step was omitted.

The last chapter is devoted to discussing some conclusions and proposing future lines of work.

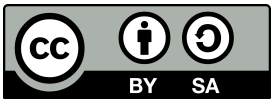
1.5. Final notes

All figures in the text have been created by the author, unless otherwise specified as a footnote in the first mention of the figure in the text.

The code for this work was developed using Scikit-learn library (Pedregosa et al. (2011), Buitinck et al. (2013)), which is released under a BSD License.

The material developed for this project can be found in the following Google Drive repository[2].

[2] If the hyperlink does not work, this is the full link: https://drive.google.com/drive/folders/12u9oWPFuA9tNz-v6xpu33By5xafh8hHp?usp=sharing



Chapter 2

Business Case Comprehension

In the previous chapter we introduced the motivation (Section 1.1), state of the art (Section 1.2) and objectives (Section 1.3), and at the end we saw an overview of the methodology that has been followed for this project realization (Section 1.4).

In this chapter we delve into the outcome of the first step defined in the methodology, which involves understanding the business case. A dataset with stock market values was provided as a study case for this project, and in the next subsections we learn the basics of stock market parameters, see what the data looks like, and define the metrics that we will use in the following steps.

2.1. Business comprehension

In the introduction above we said that "a dataset with stock market values" will be used in this project. Since our business case is related to the stock market, the first step into understanding it is by getting insights into the stock market basic terminology. This will help us understand what the provided data represents, and later on decide what can be done with it or how it can be handled.

- A *stock* is all of the parts, known as shares, into which a company's ownership is divided.
- The *stock market* is comprised of buyers and sellers of stocks. They represent ownership claims over the companies whose stocks they buy. There are multiple stock markets, usually grouped by geography or industry.
- The *open-high-low-close* (OHLC) indicators illustrate the movements of a given index for a specific day:
  - *Open* indicates the price at which transactions start.
  - *High* indicates the maximum value that the index reaches.
  - *Low* indicates the lowest value that the index reaches.
  - *Close* indicates the price the index has at the last transaction of the day. Out of the four, this one is considered the most important.
- Another two useful indicators are the *volatility* and the *volume*:

- The *volatility* measures how much the price of a given asset fluctuates around the mean price. In other words, it gives an idea of how risky it is to invest into that specific index fund.
- The *volume* refers to the amount of shares of a specific index that are traded during a given period.

There are several types of stock market indices, the three most common are:

- A *stock market index* is a standardized method that measures a stock market (or a subset of it) and gives information on its performance. They are categorized by coverage into country, regional, global, exchange-based or sector-based.
- A *commodity price index* works similarly to stock market indexes, but measures the performance of commodity assets (e.g. metals, energy or farmed goods such as wheat or soy) instead.
- A *currency index* measures the value of a given currency relative to other foreign currencies.

2.2. Data comprehension

The raw file utilized in this project contains data corresponding to 30 indices of the three types described in Section 2.1. The full list of stock market indices in this file can be seen in Table 2.1, and the list of commodity indices in Table 2.2. The only currency available in our dataset is *EUR_Curncy_Closing*, which represents the closing value of the Euro.

The exact format of the table is shown in Table 2.3. The columns correspond to the *open-high-low-close* indicators plus the volatility and volume values for each one of the indices (except for *EUR_Curncy* for which we only have the *closing* value, as noted above), while the rows correspond each to the daily value.

The daily values correspond to data for every single day from 27/06/2008 to 27/07/2018 both included (minus Saturdays and Sundays).

The idea of applying ML to this dataset emerged from the observation that certain movements in one or more indices can forecast an increase or decrease of the value of another index. This means that a ML algorithm could learn those forecasting movements from the data, and help predict the future values of a selected index.

Since the forecasting tests could not be realized for every single column due to time constraints, the *Closing* indicator was selected for three different indices, each one belonging to a different category:

- *SPX_Index_Closing* as a stock market index. This index, the Standard & Poor's 500, includes 500 leading companies in USA, including internet companies such as Alphabet (Google) or Amazon, communication companies such as Fox or Netflix, others from the Energy sector such as Exxon, financial groups such as Goldman Sachs, health care companies such as Johnson & Johnson, industrial companies including Apple, HP, General Electric, and many others. This index is very well-known because many financial products consider it.
- *TY1_Comdty_Closing* as a commodity index. This index represents a treasury note, which is a fixed-income investment issued by a country's government. In particular,

the TY1 index, short for 10 Year U.S. Treasury Note, is issued by the Treasury Department of the United States Government, with a maturity time of 10 years. It is a low-risk investment, but this also means that it has a low interest rate. This is one of the most representative long term investments.

- *EUR_ Curncy_ Closing* as a currency index. This index represents the change between Euros and Dollars.

After getting familiarized with the business domain and taking a look into the data, in the next section we decide how can the results be measured.

Index	Represents
INDU	Dow Jones Industrials
SPX	Standard & Poor's 500
CCMP	NASDAQ Composite
SPTSX	S&P/TSX Composite (Canada)
MEXBOL	Mexican stock market
UKX	FTSE 100 (UK stock market index)
CAC	French stock market index
DAX	German stock market index
IBEX	Spanish stock market index
FTSEMIB	Italian stock market index
OMX	Swedish stock market index
NKY	Japanese stock market index
HSI	Hong Kong stock market index
SHSZ300	Shanghai stock market index
AS51	Australian stock market index

Table 2.1: Stock market indices

Index	Represents
TY1	10 Year U.S. T-Note
FV1	5 Year U.S. T-Note
TU1	2 Year U.S. T-Note
RX1	Eurex Euro Bund
OE1	Eurex Euro Bobl
DU1	Eurex Euro Schatz
CO1	ICE Brent Crude Oil
CL1	Crude Oil
C'1	Corn
W'1	Wheat
S'1	Soybeans
MO1	CO2
HG1	Copper
LMAHDS03	3 month aluminium forward
LMCADS03	3 month copper forward

Table 2.2: Commodity indices

	A	B	C	D	E	F	...
1	xxx_Open	xxx_Closing	xxx_High	xxx_Low	xxx_Volatility	xxx_Volume	...
2	112,85	153,42	115,25	117,99	187,26	248816	...
3	145,7	116,51	137,3	112,56	149,74	278624	...
...	...	...	...	...	...	...	...
2632	220,52	227,07	250,22	250,04	60,18	59553	...

Table 2.3: Example of the raw data format

2.3. Metrics definition

2.3.1. Metrics available in ML

Evaluation metrics are used for measuring the quality of the chosen model. Using the right metrics is an important step for any ML project. For example, they can be used to evaluate different models' performance before choosing one model for doing further tests, and later on for evaluating how well the chosen model is working.

Many metrics have been defined for ML, specifically within the context of supervised learning, but *ad hoc* metrics can also be defined and used in certain cases. Below we define some of the most commonly used metrics in regression and classification:

1. Regression metrics

   - The *Mean-Absolute-Error (MAE)* is the absolute difference between the target and the predicted values:

     $$MAE = \frac{1}{N}\sum |y - \hat{y}|$$

     This metric does not penalize large errors and is less biased for higher values. It may not reflect properly the performance when dealing with large error values. Since it measures the absolute error, it does not account for the direction of the value.

     We will find MAE to be useful in those cases where the overall impact is proportionate to the increase in error.

   - The *Mean Squared Error (MSE)* is the squared average of the difference between the target and predicted values:

     $$MSE = \frac{1}{N}\sum (y - \hat{y})^2$$

     As opposed to MAE, it does take the direction of the value into account, and it is highly biased for high values. It does penalize large errors.

   - The *Root-Mean-Squared-Error (RMSE)* is the square root of the averaged squared difference between the target and the predicted values:

     $$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

     Since it is an extension of MSE, this metric shares many of its properties, such as taking into account the direction of the error or penalizing a large error.

However, RMSE is better at reflecting the performance when dealing with large error values.

This metric is preferred when the overall impact is disproportionate to the increase in error, as opposed to MAE.

- The *Root Mean Squared Logarithmic Error (RMSLE)* is similar to RMSE, but calculated at logarithmic scale. The formula adds 1 as a constant to both target and predicted values, since they can be 0, and log of 0 is undefined.

$$RMSLE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\log(y_i + 1) - \log(\hat{y}_i + 1))^2}$$

Like MAE, this metric does not penalize large errors, and it is commonly used if we do not want the results to be influenced by large errors. It does penalize lower errors however. Additionally, it penalizes underestimations more than overestimations.

- The R^2 *Error* tells how good the model is compared to a constant baseline:

$$R^2 = 1 - \frac{MSE_{model}}{MSE_{baseline}}$$

This metric provides a relative measure of fit, while MSE provides an absolute measure. In some situations, such as when we are looking for the relationship between different variables rather than for the prediction itself, R^2 is not a relevant metric.

2. Classification metrics

- The *confusion matrix* is a table used for determining the performance of a classifier, offering a detailed evaluation of the results. Table 2.4 is a representation of this matrix, where
  - **True Positive (TP)** represents the values that were correctly predicted as *true*.
  - **False Positive (FP)** represents the values that were predicted as *positive*, but were actually *negative*.
  - **False Negative (FN)** represents the values that were predicted as *negative*, but were actually *positive*.
  - **True Negative (TN)** represents the values that were correctly predicted as *negative*.

		Actual class	
		Positive	Negative
Predicted class	Positive	**TP**	**FP**
	Negative	**FN**	**TN**

Table 2.4: Confusion matrix

From the confusion matrix we can derive other metrics, such as:

- *Precision*: represents the ratio of the correct predictions to the total of the positive predictions:

$$precision = \frac{TP}{TP + FP}$$

- *Recall*: represents the model's ability to label correctly all the relevant cases:
$$recall = \frac{TP}{TP + FN}$$
- *Accuracy*: represents the ratio of the correct predictions to the total number of observations:
$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$
- F_1 score: takes both precision and recall into account and gives a weighted average, which can be useful when the classes are not balanced:
$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

- *Cohen's kappa coeficient* (κ) is a more robust estimator which measures the consensus between two different raters that classify the same set into mutually exclusive categories. It is defined by the following formula:
$$\kappa \equiv \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e}$$
where p_o represents the relative agreement between the raters, while p_e represents the probability of the raters agreeing by chance. If the raters disagree completely, then $\kappa = 0$, and if they both fully agree, $\kappa = 1$.

In this project two different metrics have been used:

1. The Kappa (κ) estimator, defined above, for choosing the most suitable scaler.
2. The *gain* estimator, defined below, for comparing the proportion of money earned with different indices and hyperparameters.

The *gain* estimator is an *ad hoc* estimator we have defined for representing the average return for our investment, and is described in the next subsection.

2.3.2. The gain

One of the objectives we established in Section 1.3 was to check the viability of our approach in terms of gaining and not losing. In this section we define what exactly it means to gain in the context of our business case.

A simplified description of the stock market activity could be "selling and buying stocks in order to make a profit" which is called the *capital gain*. Thus, it would be interesting to forecast how much profit would be obtained during the stock exchange activity, i.e. by either buying or selling stocks.

In short, what we want is a clear answer to the question *"How much money would we gain if we buy a certain amount of stocks and sell them later?"*, so that we can decide accordingly.

The following example will help define the formula that we will use to answer this question:

We have been following *Example_Index* and would be interested in knowing if buying one stock today and selling it in a few days would result in a gain for us. How do we express that, knowing that the *closing* value for *Example_Index* today was 40?

If we establish today as day i, we can write *Example_ Index_ Closing* as:

$$y_i = 40$$

We said above that we wanted to sell our stocks in f days. Then, we can define the absolute gain as the difference between the closing value in f days and day i. This gives us:

$$y_{i+f} - y_i$$

Now, if we express the gain as a percentage or decimal value, it makes the calculation of the profit easier to adjust when buying multiple stocks. Finally, we define it as the gain expressed in decimal form:

$$\frac{y_{i+f} - y_i}{y_i}$$

However, this market follows its own tendency: we could think that the gain is due to our ML technique while this gain is in fact due to the natural stock market growth. In order to discount this effect we substract the average increment of the index y, $\overline{y}_f$ and divide by the future f in order to get the gain per day:

$$G = \frac{\frac{y_{i+f} - y_i}{y_i} - \overline{y}_f}{f}$$

For instance if for $f = 5$ we have $y_{i+f} = 60$ and the average increment of y in 5 days is 0.1 then

$$G = \frac{\frac{60 - 40}{40} - 0.1}{5} = 0.08$$

That is, in our investment we have earned a $0.08x100 = 8\%$ per day more than the average during the 5 days.

2.3.3. The weighted gain

Our proposal for investing is the following one:

- If our ML model indicates that there is an outlier for future f and index I, then buy I stocks and sell them in f days.
- Otherwise, do not invest.

One issue with the previous gain definition is that it does not consider how many operations are carried out. That is, maybe the gain is high for future $f = 5$, but the method only considers one outlier per year. In order to measure the gain in a period of time we must consider the number of outliers per unit of time. This gives raise to the definition of weighted gain:

$$G = (\frac{y_{i+f} - y_i}{y_i} - \overline{y}_f) \times p_o$$

where p_o is the proportion of outliers per day proposed by the model. Hence, the gain measures the daily return during an operation, while the weighted gain indicates the average daily return not constrained to the operations time.

In general the gain is a good measure and is used in this work unless the contrary is stated. However in certain circumstances it is useful to compare the gain of different methods during a period of time, and in this case the weighted gain will be employed.

Chapter 3

Data Preprocessing

In the previous chapter we saw what the raw data file looks like. In this chapter we learn about the prerequisites the data needs to fulfill and the different transformations it needs to go through in order to make it ready for the ML training step.

3.1. Stationarity

A common assumption when working with time series is that the provided data is stationary. We can define stationarity as "the property of having the mean, variance and autocorrelation be constant over time". This is important, since the prediction values should not depend on the specific point of the time series. We can see a graphic example of each type in Figure 3.1[1].

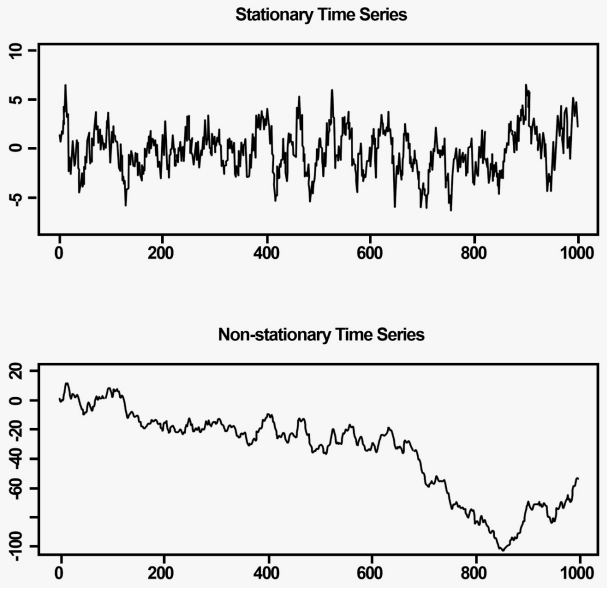


Figure 3.1: Examples of stationary and non-stationary time series

Methods such as Seasonal ARIMA (SARIMA, Hyndman and Athanasopoulos (2018)) include a local factor that allows working with non-stationary data, but in ML this factor

[1]Source: Stationarity of a time series models (O'Reilly)

is not taken into account. For this reason, the first preprocessing step was to determine whether our dataset contains stationary or non-stationary data.

Different methods, commonly known as *unit root tests*, can be used to check whether a time series is stationary or not. Some of the most well-known tests are listed below:

- *Augmented Dickey-Fuller* (ADF)

  This is one of the most widely used unit root test. The test uses as null hypothesis (H_0) the idea that a root unit is present, and the alternative hypothesis (H_a) is stationarity or trend stationarity.

- *ADF-GLS*

  This is a variation of ADF more specific to economic time series. First, it applies some modifications to the data (de-means it in order to estimate the parameters that are deterministic) and then it applies ADF.

- *Phillips-Perron* (PP)

  This is a good alternative to ADF, however it seems to have a worse performance in finite samples. It uses the same null and alternative hypothesis as ADF.

- *Kwiatkowski–Phillips–Schmidt–Shin* (KPSS)

  This test has the particularity of having the null and alternative hypothesis inverted. That is, H_0 is stationarity and H_a is unit root. It is very useful as a complement to the other tests.

3.1.1. Stationarity in our dataset

After performing preliminary tests with all of the aforementioned methods (using the implementations provided by the *ARCH* library for Python (Sheppard et al. (2020))), and obtaining similar results, ADF was finally chosen for stationarity testing.

From the result report we specifically looked at the p-value, which represents the probability of H_0 being correct. In general, a threshold value between 0.01 and 0.05 is set for the p-value, below which we can discard H_0. Discarding H_0 means that we accept H_a and, in the case of ADF, this means that we assume stationarity.

The stationarity analysis was performed on every single column. The columns were grouped by indicator type (*open-high-low-close*, volume, volatility), because of the assumption that they would behave similarly.

The average p-value for each type of column can be seen in Figure 3.4, and in Figure 3.3 we can see the individual p-values plotted for each index. On the X axis we have the columns, which correspond to the different indices, while on the Y axis we have the p-value.

In Figure 3.4 we can see that *open-high-low-close* columns have an average p-value of 0.4; additionally, if we look at the first four graphs in Figure 3.3 we can see that most plotted points are scattered above the aforementioned threshold. Thus, we can conclude that these columns are, in general, not stationary.

Alternatively, the average p-value for both volume and volatility columns in Figure 3.4 is below the threshold, and in the last two graphs in Figure 3.3 we can see how almost all of them are very close to 0, indicating that these columns are stationary.

For further processing, a possibility would be discarding the non-stationary columns and working with the volume and the volatility exclusively. However that would mean limiting ourselves to a third of the provided data and losing potentially interesting information.

We did not want to assume this limitation; this means that, as the first preprocessing step, we needed to convert the non-stationary columns into stationary.

One of the easiest and most common techniques for this is differencing. In the following section we will see that the stationarity step is covered by our proposal of data transformations, which includes differences.

Avg Open p-value: 0.40389912714896425
Avg Close p-value: 0.4005236698273309
Avg High p-value: 0.4225705719806017
Avg Low p-value: 0.40692164712904194
Avg Volatility p-value: 0.0032738005293642173
Avg Volume p-value: 0.00746837759208982

Figure 3.2: Average p-value per type of column, calculated with ADF

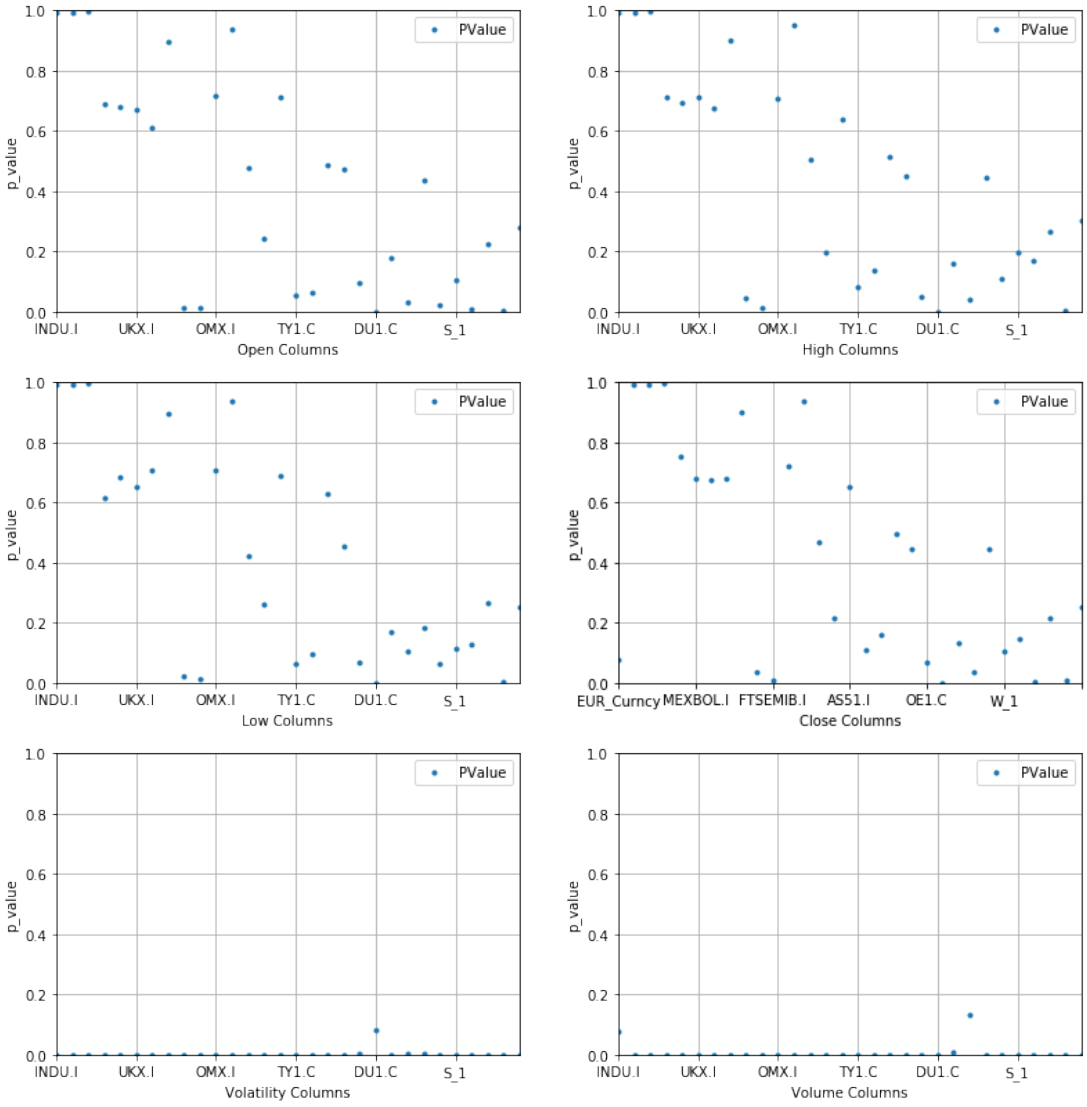


Figure 3.3: Graphic plots of the p-values per type of column

3.2. Data preparation for supervised learning

The goal of supervised learning is finding a function that, given input-output pairs as training examples, maps the input to the output. Afterwards, the learned function can be used for mapping new inputs. For a more formal definition, let X be the input vector and y the output (also known as *label*), we seek to find the function f so that:

$$y = f(X)$$

In order to find this function, we train the chosen algorithm with our (X_i, y_i) pairs, and as a result we obtain a model.

In general ML, the order in which these pairs are given for training is not relevant. However this is not the case when working with time series, since the data is ordered in time. Moreover, we typically want to learn from the previous values, which would require using multiple vectors $(X_i, X_{i-1}, X_{i-2}...)$ to obtain y_i, and this is not possible out of the box.

The solution is creating new rows by combining several consecutive rows and thus giving the ML method the opportunity to infer a future label from several past features. In particular, first we group together all the *past data* vectors we want to use for training, and then we link them with the *future* we want to predict from them. For consistency, we define two parameters:

- $p \in \mathbb{N}$: represents the past rows we want to include.
- $f \in \mathbb{N}$: represents what future value we want to predict.

Additionally, we can select which columns we want to use as input and output. We will refer to them as:

- *Xcols*: represents the list of columns we want to use as input.
- *ycol*: represents the column we want to use as output.

In the following subsections we explain in detail the grouping process and we illustrate it with a smaller scale example, using Table 3.1 as our example of a raw datafile.

	A	**B**	**C**
0	10	11	12
1	20	21	22
2	30	31	32
3	40	41	42
4	50	51	52
5	60	61	62
6	70	71	72
7	80	81	82
8	90	91	92
9	100	101	102
10	110	111	112

Table 3.1: Raw file

If we want to predict the value of column C two days ahead in the future, from the past three days' values of columns A and B (today included), we have that:

- $p = 3$
- $f = 2$
- $Xcols = \{A, B\}$
- $ycol = C$

3.2.1. Preparation of the X columns

In order to prepare the transformation of the X columns, that is, of the *features* used in ML, we need three input parameters: the raw datafile (Table 3.1), the past p and the names of the columns of the raw datafile *Xcols* that are going to be employed. Notice that, while in supervised ML, the label to predict cannot be part of these columns. However, in our case the label column can be one of those in *Xcols*. The reason is that only the past values of the column will be selected as part of the features, while a future value will be the actual final label. In fact, if *Xcols* includes only the column label we are in a case of univariate time series (see Section 1.2.1).

In our running example, the raw file is represented by Table 3.1, $Xcols = \{A, B\}$ and $p = 3$. Then, the first aggregated row we can generate is the one that combines the rows 0, 1 and 2 from the raw file:

$$X'_0 = [A_2, A_1, A_0, B_2, B_1, B_0]$$

The value A_2 represents the value of A two units of time ago (two days in our use case), A_1 represents the value of A one unit of time ago, and A_0 represents the value of A 0 units of time ago, that is, in the present (in our case this corresponds to the value of the index after the stock market has closed in the same day we are operating). Notice that each unit of time is represented in the raw file by a new consecutive row. Thus, if we consider as present any row r of this file, then r contains the value A_0, while the previous row contains A_1, and two rows above we can find A_2.

If we continue this process with the next rows, we obtain a new table of aggregated past values. Table 3.2 and Table 3.3 show the original and the result for our example datafile, and the correspondence of the values from one to the other.

Note that the columns have been renamed so that we can still know where the data comes from: we kept the name of the column, and added "p" for past, and a number which refers to how far back in the past this column goes. Also note that the transformed table contains fewer rows due to the need to start at a point where we have the required number of past columns.

	A	B	C
0	10	11	12
1	20	21	22
2	30	31	32
3	40	41	42
4	50	51	52
5	60	61	62
6	70	71	72
7	80	81	82
8	90	91	92
9	100	101	102
10	110	111	112

Table 3.2: Raw file: selected rows for X'_0

	Ap0	Bp0	Ap1	Bp1	Ap2	Bp2
0	30	31	20	21	10	11
1	40	41	30	31	20	21
2	50	51	40	41	30	31
3	60	61	50	51	40	41
4	70	71	60	61	50	51
5	80	81	70	71	60	61
6	90	91	80	81	70	71
7	100	101	90	91	80	81
8	110	111	100	101	90	91

Table 3.3: X' columns grouped for $p = 3$

For general cases, the file transformation explained above would be enough. However, in this particular use case, we wanted to calculate the *gain* we would have after selling or buying stocks. It would make sense, then, to train the model with the daily increment data

instead of the original values, so that we can predict when this increment will increase or decrease.

For this reason, during the column aggregation step, we also calculate the daily increment by using the formula explained in Section 2.3.2. For example, for the first row of A_{p0} we would calculate the new value as follows:

$$\frac{A_3 \ - \ A_2}{A_2} = \frac{40 \ - \ 30}{30} = 0.333$$

In Table 3.6 we can see the result of applying this formula to all rows for each of the columns in *Xcols*. Note that, as we do differences between two rows, the resulting file will have one less row than if we just aggregate columns, as we need to start one column further in order to be able to apply the gain formula (see how Table 3.3, with only the aggregation, has nine rows, as opposed to Table 3.6, which only has eight rows).

	Ap0	**Bp0**	**Ap1**	**Bp1**	**Ap2**	**Bp2**
0	0.333	0.323	0.500	0.476	1.000	0.909
1	0.250	0.244	0.333	0.323	0.500	0.476
2	0.200	0.196	0.250	0.244	0.333	0.323
3	0.167	0.164	0.200	0.196	0.250	0.244
4	0.143	0.141	0.167	0.164	0.200	0.196
5	0.125	0.123	0.143	0.141	0.167	0.164
6	0.111	0.110	0.125	0.123	0.143	0.141
7	0.100	0.099	0.111	0.110	0.125	0.123

Table 3.4: X' with the daily increments calculated

3.2.2. Preparation of the y column

In order to prepare the transformation of the y column, we also need three input parameters: the raw datafile (Table 3.1), the future f and the column we wish to predict at future f, *ycol*.

In the example above we established that $f = 2$ and $ycol = C$. As we only have one column, there will be no aggregation here, but rather an offset of the column C, since the first day we could theoretically predict is C_2 (however, we will see in the next subsection that this is not the case).

Like we did in the case of the X columns, we also rename the resulting column by following a similar pattern: to the name of the columns we add "f" for future and a number representing the value we assign to our f parameter.

Similarly to how we applied the incremental transformation to the X columns, we do the same here but with a difference: instead of calculating the daily increment, we calculate the increment between "today" and the future. This will give us the following formula for calculating the first row of C_{f2}:

$$\frac{C_2 \ - \ C_0}{C_0} = \frac{32 \ - \ 12}{12} = 1.667$$

In Table 3.7 we can see the full C_{f2} column after applying the offset along with the incremental transformation. Note that the table has only nine rows, as we have a two-day offset.

3.2.3. Bringing it together

Now that we have generated both X and y, we need to see how they relate to each other.

As we can see in Table 3.5, with $p = 3$ and $f = 2$, the first value we can predict is C_5 (in blue). When joining both tables, we will have to discard the previous values of the y table (grey cells in column C). Looking at the other extreme, in order to predict C_{10} (in violet), we need rows 5 to 7. Those will be the last rows from the X' table that will be included in the joint (X', y') table, and the following rows (grey cells in columns A and B) will not be part of the final dataset.

	A	B	C
0	10	11	12
1	20	21	22
2	30	31	32
3	40	41	42
4	50	51	52
5	60	61	62
6	70	71	72
7	80	81	82
8	90	91	92
9	100	101	102
10	110	111	112

Table 3.5: Relationship between past and future

In the tables below we can see the values with the extra transformations applied (Tables 3.6 and 3.7), and under we can see the final table, ready for ML training (Table 3.8).

	Ap0	Bp0	Ap1	Bp1	Ap2	Bp2
0	0.333	0.323	0.500	0.476	1.000	0.909
1	0.250	0.244	0.333	0.323	0.500	0.476
2	0.200	0.196	0.250	0.244	0.333	0.323
3	0.167	0.164	0.200	0.196	0.250	0.244
4	0.143	0.141	0.167	0.164	0.200	0.196
5	0.125	0.123	0.143	0.141	0.167	0.164
6	0.111	0.110	0.125	0.123	0.143	0.141
7	0.100	0.099	0.111	0.110	0.125	0.123

Table 3.6: X' with discarded rows

	Cf2
0	1.667
1	0.909
2	0.625
3	0.476
4	0.385
5	0.323
6	0.278
7	0.244
8	0.217

Table 3.7: y' for $f = 2$

	Ap0	Bp0	Ap1	Bp1	Ap2	Bp2	Cf2
0	0.333	0.323	0.500	0.476	1.000	0.909	0.476
1	0.250	0.244	0.333	0.323	0.500	0.476	0.385
2	0.200	0.196	0.250	0.244	0.333	0.323	0.323
3	0.167	0.164	0.200	0.196	0.250	0.244	0.278
4	0.143	0.141	0.167	0.164	0.200	0.196	0.244
5	0.125	0.123	0.143	0.141	0.167	0.164	0.217

Table 3.8: Final table after all preprocessing steps

3.3. Size of the generated files

If we take a look at the resulting table, we will find that it has $p + f$ fewer rows than the original file. However, it has $p \times |Xcols|$ more columns.

This means that, if we increase f while keeping p constant, we will have slightly smaller files (see Table 3.9). However, increasing p while keeping f constant (see Table 3.10) or increasing both at the same time, will have a great impact on the size of the file and on the efficiency of the ML methods.

f	# **rows**	# **cols**	**size (KB)**
1	2625	906	46.789
2	2624	906	46.770
3	2623	906	46.752
4	2622	906	46.733
5	2621	906	46.715
6	2620	906	46.696
7	2619	906	46.678
8	2618	906	46.659
9	2617	906	46.641
10	2616	906	46.623

Table 3.9: File properties for $p = 5$

p	# **rows**	# **cols**	**size (KB)**
1	2625	182	9.404
2	2624	363	18.743
3	2623	544	28.074
4	2622	725	37.398
5	2621	906	46.715
6	2620	1087	56.025
7	2619	1268	65.328
8	2618	1449	74.623
9	2617	1630	83.911
10	2616	1811	93.192

Table 3.10: File properties for $f = 5$

3.4. Stationarity revisited

Finally, we performed the stationarity test on the new file to check that it is indeed achieved by the transformations. In Figure **??** we can see that the average p-values for the volume and volatility are still lower than the threshold, which means that stationarity is preserved by our transformation. Moreover, now the p-values for the *open-high-low-close* columns are also very low, which means that they are now stationary as well. In Figure 3.5 we can also see that plotted values per column are close to 0 for all six types of columns.

Avg Open p-value: 5.149594582984639e-17
Avg Close p-value: 1.2851526710460355e-14
Avg High p-value: 5.5440240417826706e-14
Avg Low p-value: 2.8827014823822843e-16
Avg Volatility p-value: 8.970957744170934e-20
Avg Volume p-value: 7.808701795809326e-06

Figure 3.4: Average p-value per type of column, calculated with ADF

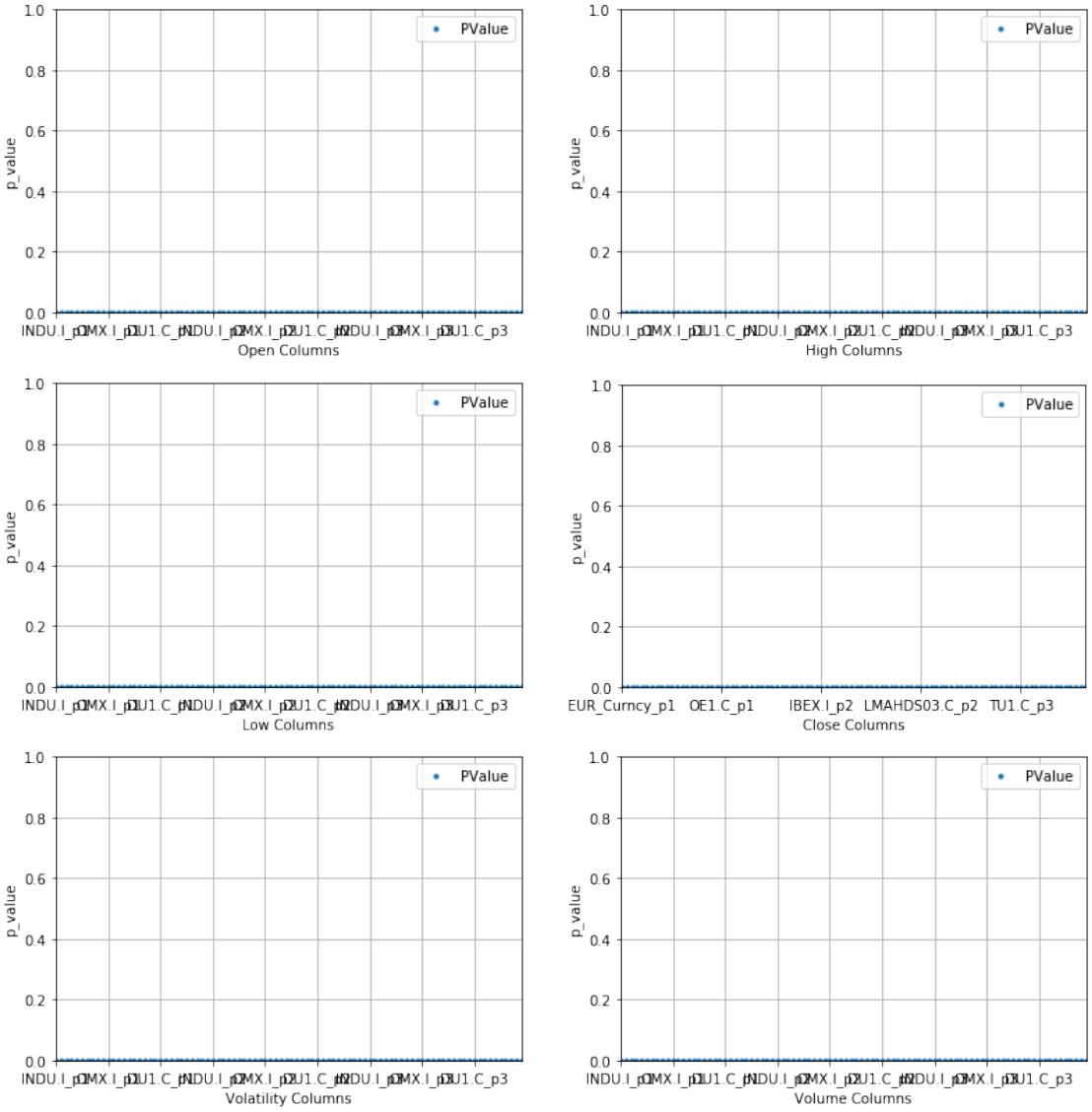


Figure 3.5: Graphic plots of the p-values per type of column after the transformations

Chapter 4

Training Time Series With Outlier Values

In the previous chapter we prepared the data for the ML step. In this chapter we define what exactly the "ML step" involves: while training the chosen ML algorithm with our prepared dataset is, effectively, one step, a few other aspects have to be determined. In this chapter we see the additional steps needed to complete the evaluation schema before training the ML algorithm of choice with a time series.

4.1. Outlier detection

As we said in Section 1.2.1, this project's focus is on detecting outliers in time series, and we explored the different types of outliers and detection techniques proposals for each type. Also, in Section 1.3.1 we limited the scope of this project to univariate global outliers in multivariate time series.

We also mentioned that we will detect the outliers by using a model-based technique: we chose using the mean and the standard deviation in order to decide whether a value was an outlier or not, and also introduced an α parameter for setting the threshold.

The α parameter ($0 < \alpha < 1$) is what allows us to decide if we want more or less outliers. The closer to 0 we get, the lower the threshold, and vice-versa.

The process to calculate outliers was the following:

1. We calculate the standard deviation (std) and the mean for the dataset.

2. We established the threshold as: $value > mean + (\alpha \cdot std)$.

3. We can look for outliers either above the threshold, below (in this case the threshold is $value < mean - (\alpha \cdot std)$), or both. A new column, *label*, will hold the values corresponding to the result of comparing each element of our y column to the threshold. This will be our new y' column which will be actually used in the next step.

It is important to mention that the rows labelled as outliers are, or should be, a minority by definition of outlier. The amount of detected outliers can be tuned by using different values of α. This can have different implications on the final results, which will be discussed in the next chapter.

4.2. Train/test sets definition

In Section 3.2 we mentioned that it is important to take the time property of the data into account, and applied column grouping transformations in order to ensure that this condition is fulfilled.

While this solution solved the problem of considering multiple past values when training the model, we still need to ensure that we keep the data in its chronological order for testing, since we need to test the model with a future value.

Note that the standard train/test split method[1] employed usually in ML must be discarded because it does not keep the chronological order while splitting the dataset. In fact, this method chooses the train and test sets randomly, and therefore we might end training with values from the future and testing values of the past.

Thus, we looked at the alternative specially made for time series, offered by Scikit-learn, called TimeSeriesSplit[2]. This is the description provided in the documentation:

> Provides train/test indices to split time series data samples that are observed at fixed time intervals, in train/test sets. In each split, test indices must be higher than before, and thus shuffling in cross validator is inappropriate.

This method creates incremental splits where the train set grows in each iteration, and the test values are the following values. In Figure 4.1 we can see an example of how it splits the dataset into train/test subsets for $n\_splits = 12$:

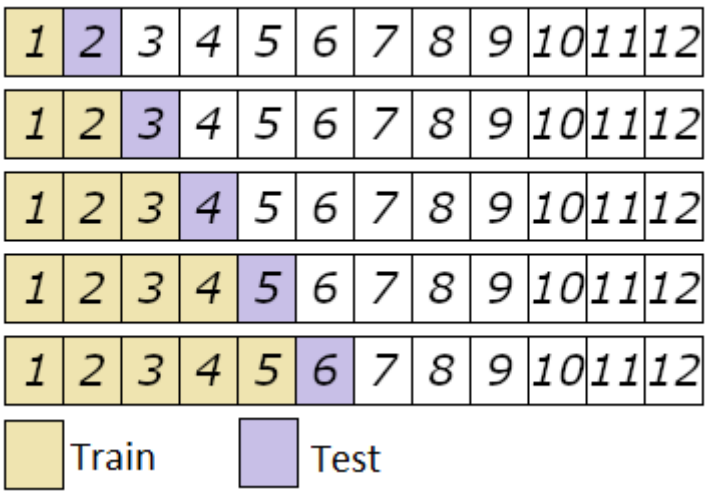


Figure 4.1: How Scikit-learn's TimeSeriesSplit works

While this seems to be exactly what we need, we have found that it did not work in our case for a few reasons:

1. As mentioned in the previous chapter, we want to predict whether the value in f days will be an outlier or not. If we use the values between the current day and the day we want to predict, we would be training the model with data that is yet unavailable. To better explain this, let's take another look at Table 3.8:

[1] For example, Scikit-learn's test_train_split
[2] Scikit-learn's TimeSeriesSplit

	Ap0	Bp0	Ap1	Bp1	Ap2	Bp2	Cf2
0	0.333	0.323	0.500	0.476	1.000	0.909	0.476
1	0.250	0.244	0.333	0.323	0.500	0.476	0.385
2	0.200	0.196	0.250	0.244	0.333	0.323	0.323
3	0.167	0.164	0.200	0.196	0.250	0.244	0.278
4	0.143	0.141	0.167	0.164	0.200	0.196	0.244
5	0.125	0.123	0.143	0.141	0.167	0.164	0.217

In this table we have that $p = 3$ and $f = 2$. If we use t (today) for the day we know the value of $Cf2$ for a row r we will have, by definition, that the X values of row r include:

a) $t - 2$ (2 days ago, $Ap0$ and $Bp0$)

b) $t - 3$ (3 days ago, $Ap1$ and $Bp1$)

c) $t - 4$ (4 days ago, $Ap2$ and $Bp2$)

On the same row r we also have the y value, which is column C for today.

Supposing that we use rows 0, 1 and 2 for testing and then predicting row 3, as *TimeSeriesSplit* suggests. In a real case, where we have the complete data for row 2 available for training, it means that we are actually on day t, in which we know the value 0.385 for $Cf2$.

But, by the same reasoning, we will realize that row 3 will have in its $Cf2$ column the increment that will happen between tomorrow $(t+1)$ and yesterday $(t-1)$. This is not our goal: we wanted to see the increment two days ahead. That is $t+2$, which is available in row 4.

This means that we need to test with row 4, which does contain the corresponding increment between $t + 2$ and today.

Also note that this is completely possible since row 4, corresponding to $Cf2$ on day $t + 2$, will have as features:

a) $t + 2 - 2$, that is, t in $Ap0$ and $Bp0$

b) $t + 2 - 3$, that is, t-1 in $Ap1$ and $Bp1$

c) $t + 2 - 4$, that is, t-2 in $Ap2$ and $Bp2$

We can use these values, since they are not part of the future, and the value we want to predict is, as we saw, $t + 2$.

In conclusion, we need to leave an $f - 1$ days gap between the last train row, and the row for testing.

2. Additionally, depending on how many n_splits we chose, the test set would have multiple values, out of which some would not be matching the aforementioned gap during the training phase, skewing the results.

3. Lastly, with the low percentage of outliers in our dataset, the likelihood of starting with train splits containing only elements of the non-outlier class is high. This would lead to a failure in the training step since there is only one class to predict.

This led to the implementation of our self train/test splitting class, which offers two methods:

1. The first method returns train/test splits on a sliding window and it can be tuned with the following parameters:

   - f: it is used for calculating the corresponding test value for each train split
   - k: specifies how many rows we want for the training splits
   - *step*: specifies how much to jump for the next split

   Figure 4.2 shows an example of splits for $f = 3$, $k = 3$ and $step = 1$:

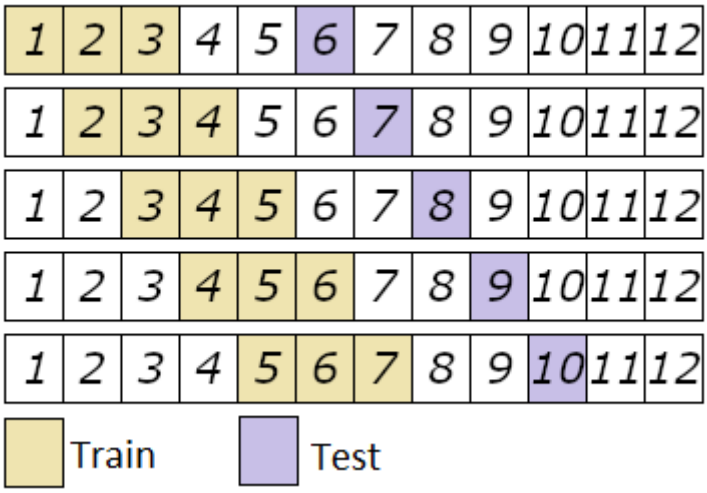


Figure 4.2: Example of how self-split works for $f = 3$, $k = 3$ and $step = 1$

Notice that, as opposed to *TimeSeriesSplit*, this method does preserve the gap between the last value used for training, and the test value; and that it only uses one test value per train set, which is the one value that corresponds to the training, as we explained above.

If left as is, this method works fine for regression, but if we want to classify outliers we might still run into the one-single-class-in-a-split issue. This is solved by the second method:

2. The second method, in addition to the previous parameters, takes a new one, *minclass*, which specifies the minimum amount of representatives of each class we want in our slices. It works in two steps:

   a) It calculates the first slice by increasing the window provided by the k parameter until the minimum amount of class representatives condition is satisfied, and with this we identify the first value we can predict.

   b) Now that we have the test value for the first slice, we can move forward and calculate the following test values by using the *step* parameter. Since we still need to maintain the gap for the train set, while also ensuring the number of class representatives, we can extend the slice backwards, as from the first step we have the guarantee that we will find at least the minimum amount of elements of each class. This results in more dynamic splits and ensures it will always be possible to train with this sets, as we will never run into the one class problem.

Figure 4.3 shows how, despite $k = 3$, the first slice has an extra element, and marks the first value we can effectively predict. The following train splits are calculated by first taking the number of rows specified by k (marked in red) and, if this piece does not contain the required number of elements of each class, it gets expanded to the left until it finds them (marked in blue).

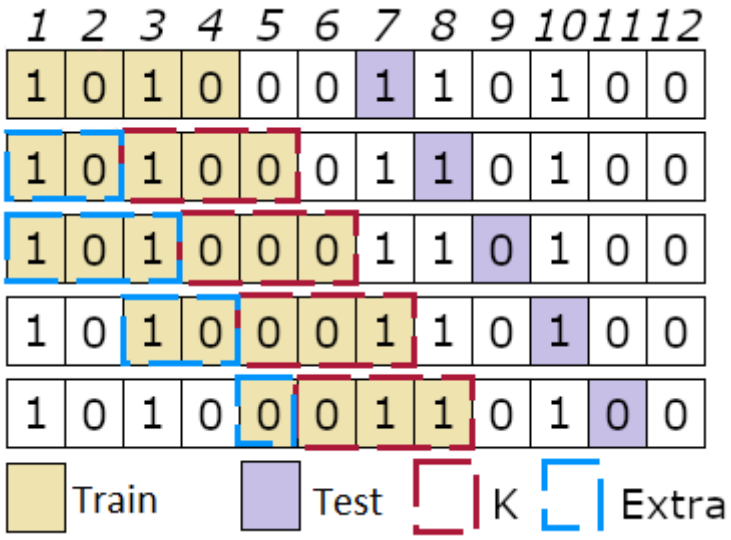


Figure 4.3: Example of how the second self-split works for $f = 3$, $k = 3$, $step = 1$, and $minclass = 2$

4.3. Dataset balancing

A common issue in classification problems is having suspiciously high accuracy from the first tests. But when we take a closer look at the results, we realize that our classifier predicts the same class most of the times, and the accuracy comes from the fact that our dataset is imbalanced: it has more representatives of one class than of the other(s). Imbalanced datasets are common: rarely will we find a set with the exact same number of elements of each class "in the wild". However, when we have heavily imbalanced sets, this can lead to an *overfitting* problem: our algorithm learns to predict the most prevalent class and has a high accuracy for this specific data set, but will perform poorly on a different set.

For instance, if the goal is to distinguish between cats and dogs, and we feed the ML method mainly with photos of dogs, it would infer that "everything is a dog". This conclusion is indeed almost correct in this dataset, and metrics such as recall or precision will give very good scores. The problem arises when we try this model on a dataset that contains a comparable number of dogs and cats. Then, we would discover that our model labels almost all the cats as dogs and that therefore it does not behave as well as we thought.

In this project, where we are specifically looking for outliers, an imbalance in our dataset is guaranteed. We need, thus, to balance our dataset in order to have similar class representation. We can do this either by oversampling or by undersampling (He and Garcia (2009); Skryjomski and Krawczyk (2017)). These two techniques allow us to adjust the class distribution by removing or adding elements to out dataset, thus, making the dataset

balanced:

- Undersampling consists of removing elements of the overly represented class. It can be achieved by randomly removing samples of the majority class or by using more sophisticated techniques such as Tomek links (Tomek et al. (1976)). One drawback of undersampling is that, by removing elements, we lose information.
- Oversampling consists of adding copies of the under-represented class. One of the earliest methods proposed for oversampling is duplicating elements of the minority classes, although more complex techniques such as Synthetic Minority Over-sampling Technique (SMOTE, Chawla et al. (2002)) are also available. This technique is more widely used than undersampling.

In our approach we oversampled the dataset with the random sampling method offered by the Pandas library for Python (Reback et al. (2020)).

4.4. Scaling

The dataset might have values of different magnitudes for each one of its multiple variables. To make sure that the algorithm is not giving more weight to some features over others, it can be necessary to scale the data so that it is in the same range of values.

The most suitable scaling algorithm will vary depending on the specific dataset. Table 4.1 shows the list of scalers[3] that have been considered in this work. These scaling algorithms were tested against unscaled data in order to find which one was best suited for the particular case at work.

Scaling algorithm	Description
Standard scaling	Standardize features by removing the mean and scaling to unit variance.
Min-max scaling	Transform features by scaling each feature to a given range.
Max-abs scaling	Scale each feature by its maximum absolute value.
Robust scaling	Scale features using statistics that are robust to outliers.
Normalizer	Normalize samples individually to unit norm.

Table 4.1: Short description of the scaling algorithms used in this work

[3]The descriptions are taken from the official documentation of Scikit-learn

Chapter 5

Use Case Results

In this chapter we present the results of the different experiments that we performed with our use case (the stock market prediction) in order to assess the techniques proposed in Chapters 3 and 4.

5.1. Parameters

In the two previous chapters we have mentioned different parameters (more formally *hyperparameters* in the context of ML) whose values can be adjusted for testing the framework. Additionally, each parameter can take a variable amount of values. Hence the combination of different parameters leads to a vast amount of choices. In ML, usually a technique such as *cross validation* is employed to improve the efficiency in terms of time when choosing the best combination. This technique cannot be employed in our setting due to the temporal relationship among data and, since it is not feasible to test all possibilities, the values for each parameter were reduced to a limited subset and some arbitrary order, presented below, was chosen in order to choose the best value for each one of them.

Below we compiled these parameters in a list, with some notes on how some of them were handled:

1. Data selection:
   - p: number of past values to use.
   - f: unit of times in the future of the value we want to predict.
   - *Xcols*: the columns to use for training, the features.
     For preprocessing we used all 181 columns from the raw file to generate the new files. However in the later stages we performed tests with different subsets of the columns, since using all of them can give problems of lack of memory and it is very slow. This is discussed in Section 5.5.
   - *ycol*: the column to predict.
     As stated in Section 2.2, the selected indices for this work are the *closing* values for columns *SPX_ Index* (Standard & Poor's 500), *TY1_ Comdty* (10 Year U.S. T-Note) and *EUR_ Curncy* (change Euro-Dollar). We will be referring to them by their short index name (SPX, TY1 and EUR respectively).
2. Outliers:

- α: outlier threshold adjustment.

  Unless otherwise specified, the tests were performed with $\alpha = 0.5$. In Section 5.7 we cover the effect that this parameter can have on the results.

- *above*: outliers above or below the threshold.

  In this work we only consider outliers *above* the threshold, although our experiments show similar figures in both cases.

- *center*, *dispersion*: Model to be used for generating outliers.

  We consider outliers of the form $center + \alpha \times dispersion$ and suggest two possibilities:

  - *center*: the arithmetic mean, *dispersion*: standard deviation (SD).
  - *center*: the median, *dispersion*: median absolute deviation (MAD).

3. Train/test splitting.

   These is in fact a set of three parameters:

   - k: minimum train slices size
   - *step*: distance between slices
   - *minclass*: minimum number of representatives of each class

4. Scaling method.

   The scaling algorithm was the first parameter for which we wanted to pick a value, so we did an analysis in order to see which one was more suitable. The results are discussed in Section 5.3.

5. ML algorithm.

   The ML classification method chosen. In Section 5.8 we cover other ML alternatives.

Unless stated the contrary, in the rest of the chapter we have:

- *Xcols*: the subset of the columns including the *High* indicator (that is, the maximum value reached that day).
- *ycol*: the *SPX Index* at closing time.
- α: 0.5.
- *above*: true (that is, only outliers over the center).
- *center*: arithmetic mean
- *dispersion*: SD
- k: 70.
- *step*: 1.
- *minclass*: 2.
- *scaling*: no scaling
- *method*: Logistic Regression.

The default metric is the gain defined in subsection 2.3.2 but expressed as percentage instead as proportion of one, although occasionally we employ the *weighted gain* (subsection 2.3.3) and the kappa score κ.

Before start choosing the results for different values of the parameters we introduce some simple methods that will serve as baseline predictions.

5.2. Baseline predictions

In order to have a base comparison point, we performed some preliminary tests with three predictors. Figure 5.1 shows the baseline predictors results, where the X axis represents the f value, and the Y axis represents the capital gain. The predictors are explained below:

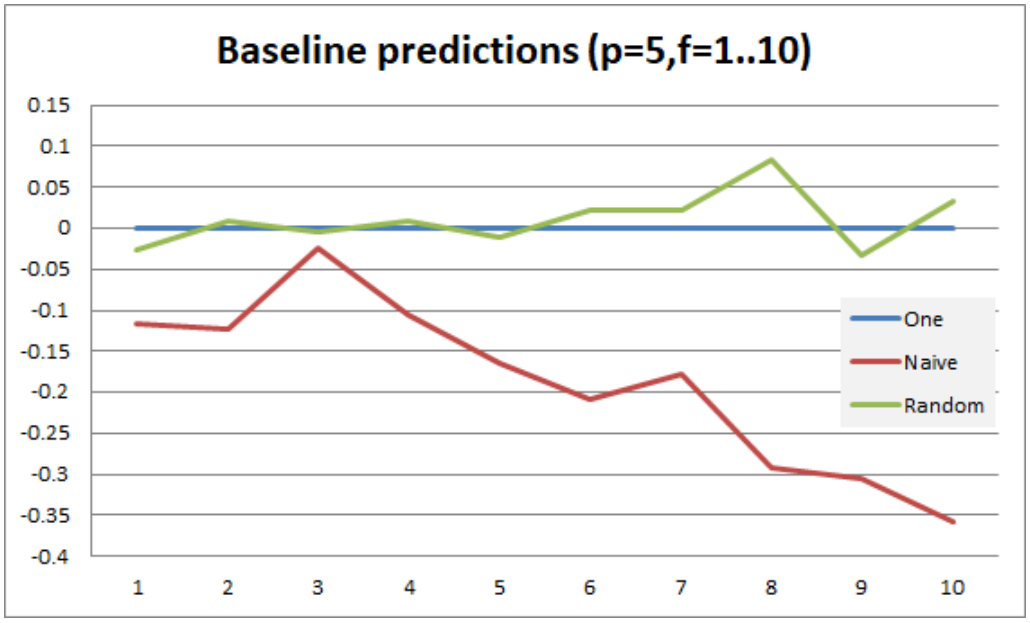


Figure 5.1: Baseline predictions

1. *One*

   This predictor is the equivalent of buying on the first day, and selling at the end (in fact it asks us to buy every day and sell after f days, but, ignoring the operations fees, this is almost equal to buy the first day and selling the last one). With this strategy, we would only gain the amount relative to the stock market growth, but our capital gain metric, which corrects this effect, would be 0. The only purpose of this predictor is to check that the gain metric behaves as expected.

2. *Naïve*

   This predictor is a standard baseline method which predicts as next value the last value. Although this can make sense in regression, it is not suitable for classification. As expected, this strategy performs poorly.

3. *Random*

   As its name implies, this predictor returns a value at random. The long-term average gain with this method will be negative.

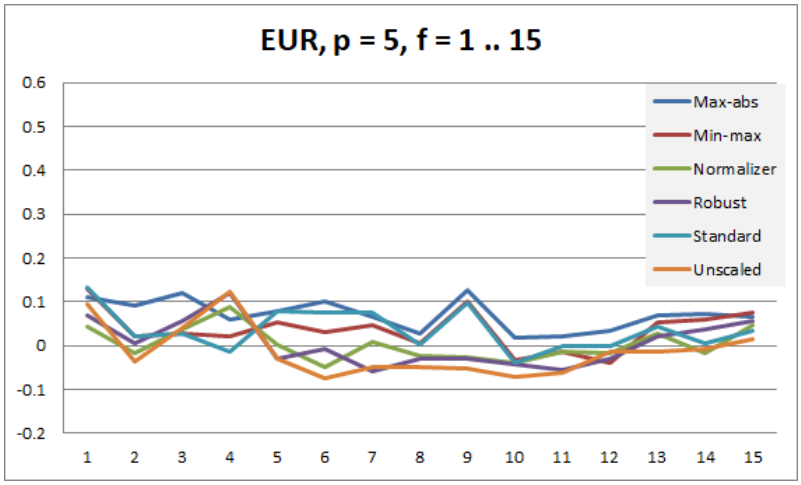


Figure 5.2: Scaling test result for EUR with $p = 5$

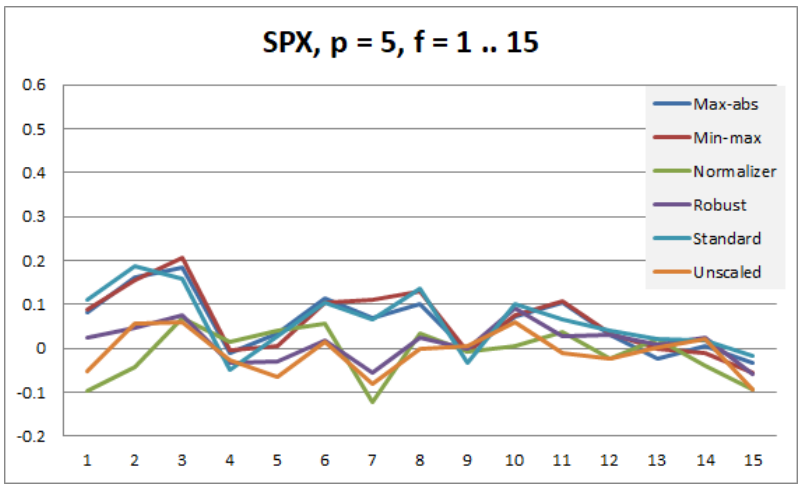


Figure 5.3: Scaling test result for SPX with $p = 5$

5.3. Choosing a scaling algorithm

In order to choose the scaling algorithm we evaluated each one of the scalers described in Table 4.1 (along with unscaled data), for each of the three indices described in Section 2.2. We used the κ estimator (explained in Section 2.3) for comparing the results.

If we take a look at Figure 5.2, where we tested the scalers with the EUR index for $p = 5$, the Max-abs scaler seems to stand out over the other scalers. But if we look at Figure 5.3, where we tested with the SPX index for the same p value, we can see that Max-abs, Min-max and Standard scalers perform somewhat better than the Normalizer and Standard scalers, as well as the unscaled data. Even so, neither stands out on its own. In both figures, the X axis represents the f value, while the Y axis represents the κ value.

After having performed more tests for different values of p and also for the TY1 index (see Figure 5.4), not only did we not have a clear winner, but actually some scalers seemed to behave better or worse depending on the chosen parameters. It is also important to note that employing a scaling algorithm added a significant increase in the execution time, especially when moving toward higher values for p.

Due to these reasons, for the subsequent tests, it was decided to work with unscaled

data. This decision is also supported by the fact that, during the preprocessing stage, the extra transformations change all values into values between 0 and 1, which brings them into a similar order of magnitude.

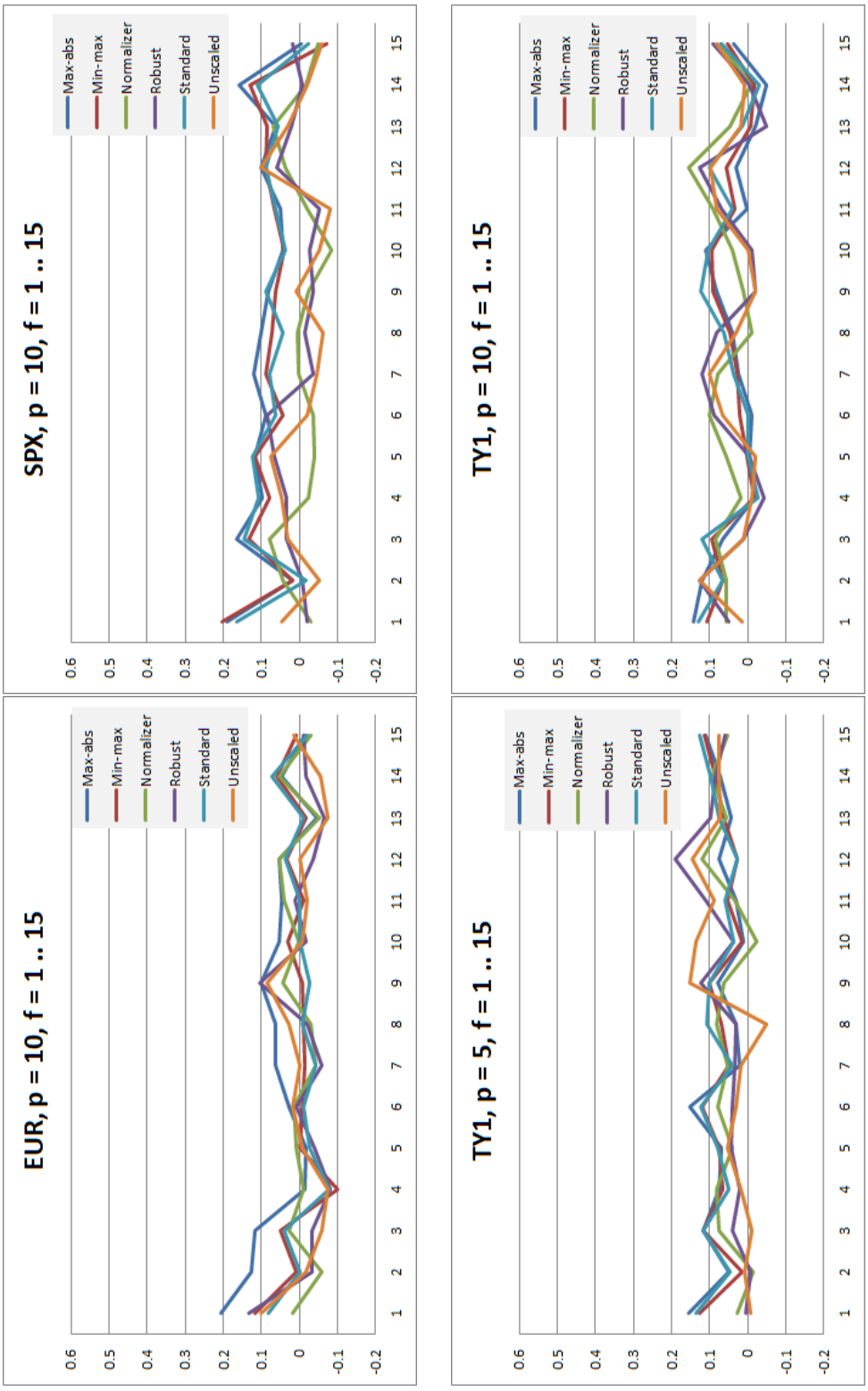


Figure 5.4: More scaling test results (EUR and SPX with $p = 10$, and TY1 with $p = 5$ and $p = 10$)

5.4. Selecting one index

Even after deciding to work with unscaled data, the run times were too high to keep working with the three indices. For this reason, we compared them in order to select the most promising one for further tuning. In Figure 5.5 we can see the results of a comparison run for past $p = 5$, where the X axis represents the values for future f and the Y axis represents the gain. This test was performed using only the *High* columns as features.

After these comparison tests, the SPX index was chosen since it seemed to have the most promising results.

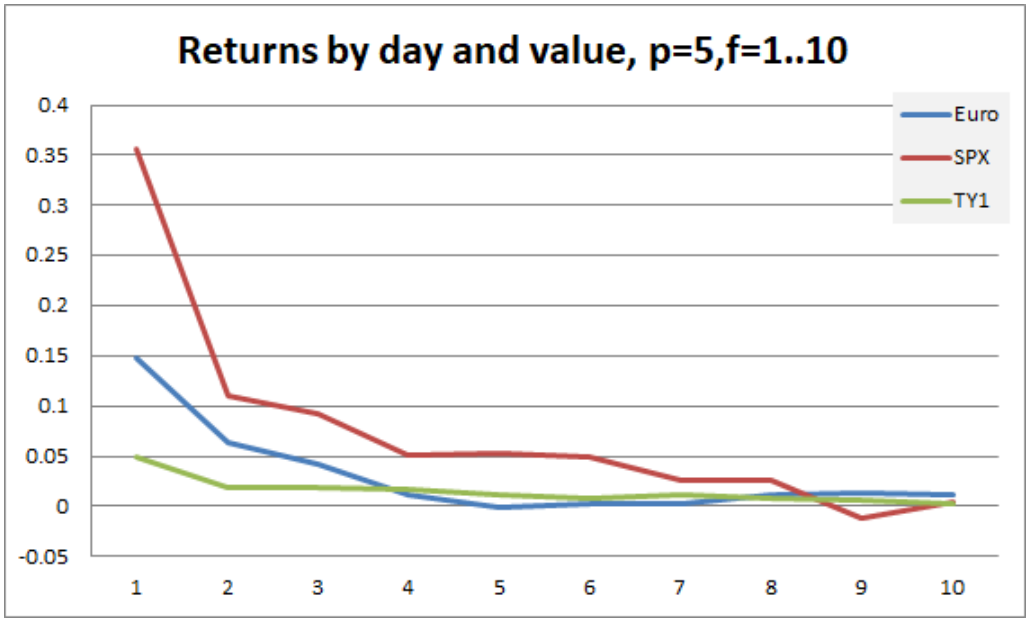


Figure 5.5: Indices comparison

Another important conclusion we can draw from the results shown in Figure 5.5 is that we meet the *forward monotony* objective established in Section 1.3. That is, given a p past value (in this case, $p = 5$), the results degrade the further away in the future we predict. We can see this happen for all three indices.

5.5. Selecting columns for training

When we described the raw data file in Section 2.2, we introduced the multiple different stock market indices contained in our dataset, and the different types of columns for each. In the same section, we also expressed the idea that "certain movements in one or more indices can forecast an increase or decrease of the value of another index".

At the beginning of this chapter we mentioned that the preprocessing step was carried out with all indices from the raw file. This allowed us to generate files with the maximum amount of information, so that we could later perform multiple tests on the same file, each one with only certain parts selected.

In order to explore the aforementioned idea, we first checked if using all columns yields some benefit over predicting, for instance, the SPX value only from it's own past results. Figure 5.6 shows that, indeed, using all columns provides much bigger gains. Using only the High columns is much worse, but still better that relying only on the SPX columns.

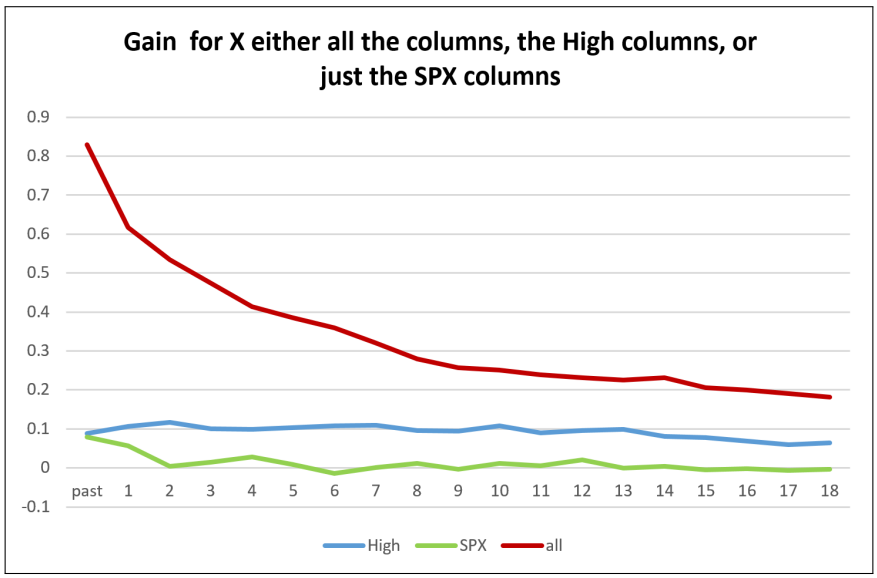


Figure 5.6: Gain for all columns, only the High columns, and just the SPX columns

It seems that the best choice would be to keep all columns. However, the Python libraries run out of memory for values of p around 15 with all columns and, even before running out of memory, the computations take too long. Thus, we state here that while using all columns would be preferable, due to our limitation of resources we needed to constrain the tests to some columns.

In order to select the better group of columns we performed multiple tests for the SPX index, each with only a certain group of columns selected. In particular, the columns were grouped by type of indicator (*open-high-low-close*, volatility and volume).

According to the results shown in Figure 5.7, using either *High* and *Low* lead to better predictions for the SPX index than using any of the other indicators, especially in a short-term timeframe. As is the case with the other graphics, the X axis represents f and the Y axis represents the capital gain.

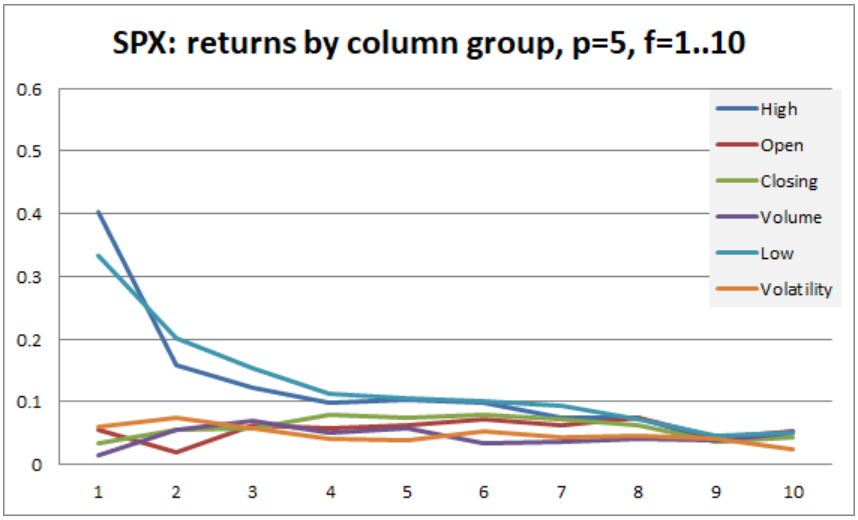


Figure 5.7: Column comparison for SPX index

5.6. Backwards monotony

In Section 1.3 we established as a goal getting coherent results for monotony, both forward and backward. In Section 5.4 we already confirmed that we have forward monotony.

Backward monotony was defined as having an improvement in the results when including more past values on each row.

In Figure 5.8 we can see the evolution of the gain as we increase the past values, for $f = 5, 10, 20$.

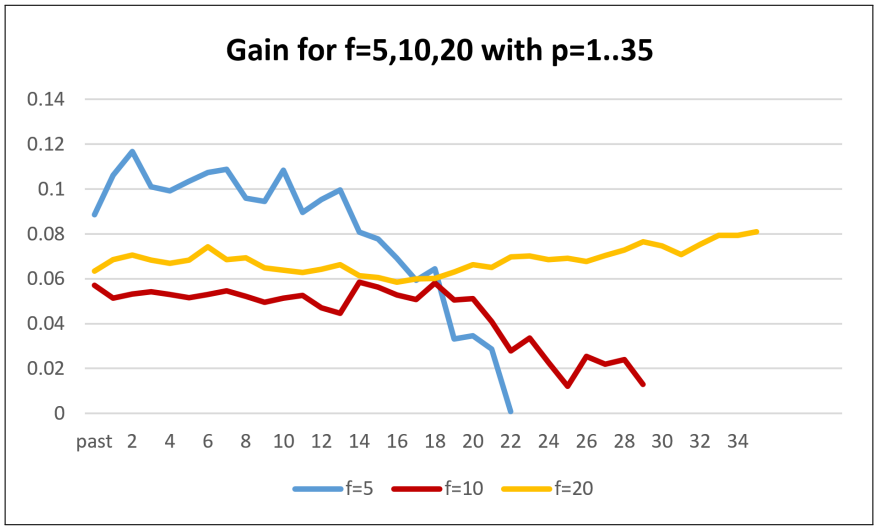


Figure 5.8: Backward monotony for $f = 20$

We observe that in the case of $f = 5$ there is no appreciable improvement when the past increases. In fact, the gain approximates zero indicating that this additional information is, for this future, too noisy. In the case of $f = 10$ we have the same effect, but the decreasing starts later, around $p = 19$, and at this point it seems to be a local maximum.

Finally, with $f = 20$ we observe our expected increase in gain when more information (more past values) is considered. The problem is that, for $p = 37$ and beyond, the Python *Scikit-learn* library stops working due to memory problems. It seems logical that the gain will also start to decrease at some point, but we cannot check this in our setting. The future work section indicates some possible solutions to this problem.

5.7. Tuning outliers by changing the value of α and the model parameters

As explained in Section 4.1, the α parameter was introduced for adjusting the threshold at which we would decide whether a value is an outlier or not.

In the stock market context, we can be looking to buy stocks of an index when we expect its value to have a significant increase in the future. (Or, alternatively, we can be looking to sell now and re-buy it at a later point if we detect a drastic decrease of its value.)

Our α parameter is how we quantify the "significant increase" in this case. Using a higher α value raises the bar at which we draw the line between outlier and non-outlier. This would result in detecting fewer outliers; that being said, the detected outliers would likely correspond to more distinctive circumstances, in which case our system would learn to recognize them better.

Back to the stock market, if we look at the gain for higher α in Figure 5.9 (SPX with *High* columns, X axis corresponds to α, Y axis corresponds to gain), it might seem that our earnings (gain, blue line in the graphic) are increasing as we increase its value.

While that is definitely not untrue, as we noted above, increasing the α parameters means that we detect *fewer* outliers, which in turn means we do fewer operations (since we decided we would only buy when the model points to a future increase).

It is in this context where the *weighted gain*, explained in Section 2.3.3 comes into play (red line in the graphic): despite having a higher return per operation, the overall daily return turns out to be lower because we do fewer operations.

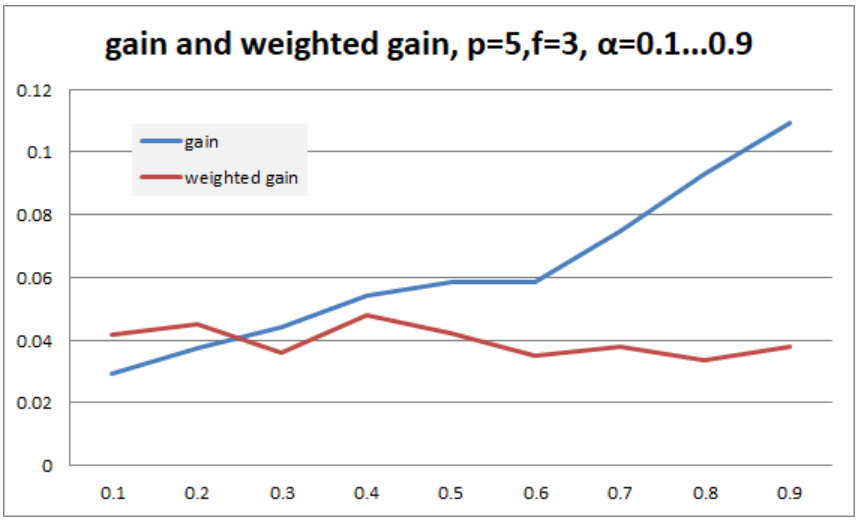


Figure 5.9: Results for different α values

The conclusion seems to be that the value α does not have a great effect on the weighted gain, although in our particular dataset slightly better results seem to be obtained for α around 0.4.

Next, we compared the weighted gain obtained when using the mean as center and the SD as dispersion metric with the outliers obtained when using the median and MAD as dispersion metric.

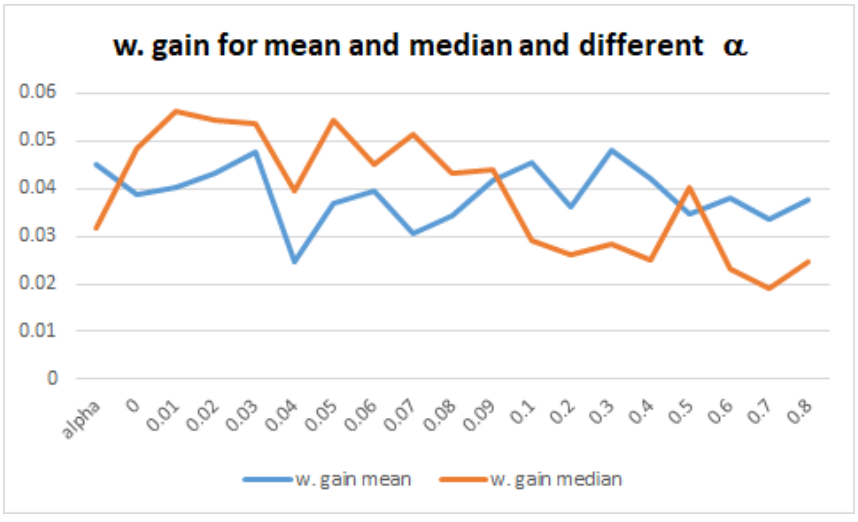


Figure 5.10: Weighted gain for median and mean, and for different α values

Figure 5.10 shows the weighted gain of both models for different α values. Notice that the x axis is split into two parts with different scales, above 0.1 and below 0.1. The result seems to indicate that the best values are obtained for $\alpha < 0.1$ and using the median and MAD. Notice however that the computation of the median is usually slower and for that reason the alternative was used for most experiments.

5.8. Testing other ML methods

As a final experiment, we wanted to evaluate how other ML algorithms would perform within the same set of values for the other parameters, as opposed to Logistic Regression. Without going into too much detail, in the list below we summarize the considered alternatives[1]:

- Quadratic Discriminant Analysis (QDA)

  A classifier with a quadratic decision boundary, generated by fitting class conditional densities to the data and using Bayes' rule.

- Random Forest

  A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

- Naive Bayes

  Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.

- Decision Tree

  Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

- Nearest Neighbors

  Neighbors-based classification is a type of instance-based learning or non-generalizing learning: it does not attempt to construct a general internal model, but simply stores instances of the training data. Classification is computed from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point.

- Support Vector Machines (SVM)

  Support vector machines are a set of supervised learning methods used for classification, regression and outliers detection. Different kernel functions can be specified for the decision function.

[1] The descriptions are taken from the official documentation of Scikit-learn

- Support vector classifier (SVC) with Radial basis function (RBF) as kernel

  The RBF kernel is a stationary kernel. It is also known as the "squared exponential" kernel.

- Linear SVM

  LinearSVC is another (faster) implementation of Support Vector Classification for the case of a linear kernel. This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.

- Neural network

  Supervised learning algorithm that learns a function $f(\cdot) : R^m \rightarrow R^o$ by training on a dataset, where m is the number of dimensions for input and o is the number of dimensions for output. Given a set of features $X = x_1, x_2, ..., x_m$ and a target y , it can learn a non-linear function approximator for either classification or regression. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers.

- AdaBoost

  An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

- Gaussian Process

  Gaussian Processes (GP) are a generic supervised learning method designed to solve regression and probabilistic classification problems. The prediction is probabilistic (Gaussian) so that one can compute empirical confidence intervals and decide based on those if one should refit (online fitting, adaptive fitting) the prediction in some region of interest.

In Figure 5.11 we can see the comparison between all these methods. The Gaussian Process seems to have by far the best results. Indeed, Gaussian Processes (Williams (1998)) are widely recognized as one of the best prediction methods for markets (Ou and Wang (2009)). However the big difference with the next method is very noticeable.

The second best method is Ada Boost. This can be due to the facility of this technique for mimicking square signals, which is the case of our outlier setting, as we will see in the conclusions. Anyway, the results of AdaBoost are similar to those of the Logistic Regression, the neural networks and the SVM methods.

A better performance was expected for neural networks. Maybe this is due to the fact that neural networks are highly configurable, and here we have used just the basic default parameters.

Before finishing the chapter, we must point out that usual techniques for tuning hyperparameters such as *cross validation* are not available in our system. This, combined with the high requirements of our algorithms in terms of time and memory, have complicated experimenting with more exhaustive combinations of parameters.

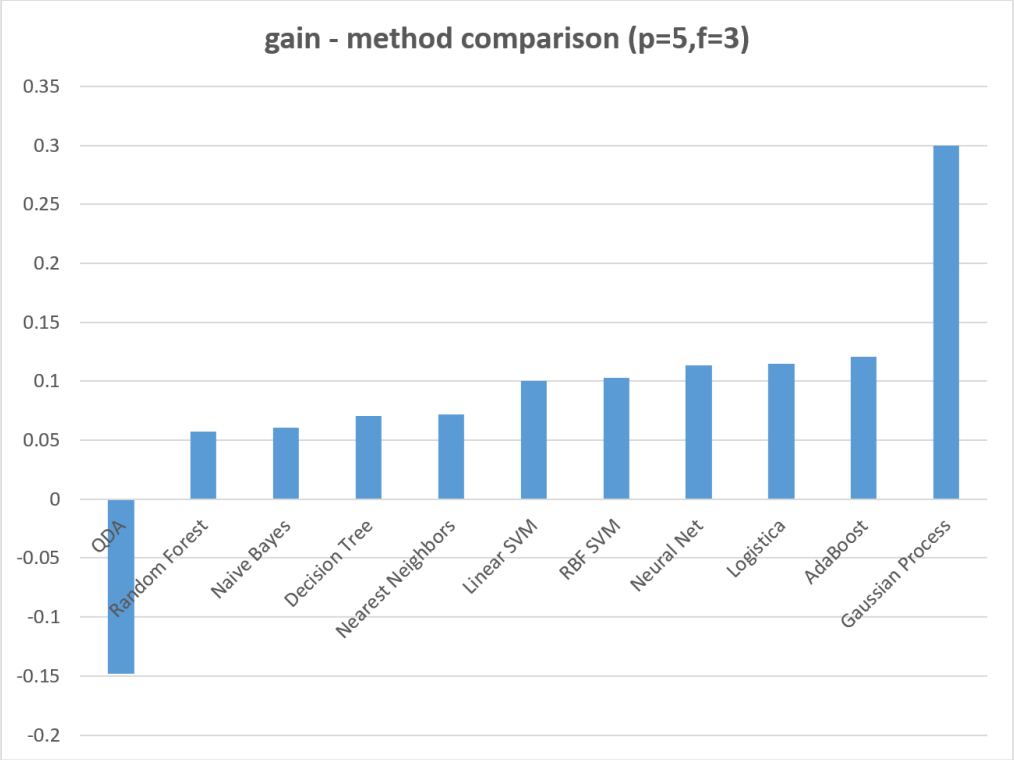


Figure 5.11: Comparison of other ML algorithms

Chapter 6

Conclusions and Future Work

In this chapter we summarize the results achieved during the work, examine if our objectives have been fulfilled and suggest some lines of future work.

6.1. Conclusions

As we have seen across the three previous chapters, the usage of time series data in the ML context is not automatic. On the contrary, it requires oversight and attention to detail in order to succeed. Otherwise it is possible, for instance, to obtain too high evaluation scores, because we have unintendedly trained with future values to predict past values, or we are testing a row that includes a label corresponding to some event that occurs before the future we wish to predict. In any case, the final effect is that, when taking the model to a production context, we will find that the evaluation scores are much worse than those predicted during the model evaluation phase.

In this work we proposed a framework for training machine learning algorithms with time series, which consists of two parts, corresponding to the first two objectives of Section 1.3:

1. A preprocessing method that combines past and future data, explained in Chapter 3. The method takes as inputs the file containing the raw data, a set of feature columns, a future f to predict, and a number of previous (or past) rows p. The result is a preprocessed file that combines each row r with the $p-1$ previous rows as features, and with the target value at row $r+f$ as label to predict. We also propose to work with increments (one day increment for the features, f days increment for the label) for our use case. Notice that combining pasts $1, \ldots, p$ and future f in one single row is important because ML methods work row by row, adjusting the internal parameters to infer the label from the features. With our preprocessing the ML method 'sees' a window of p features together with the expected value at future f.

2. A training and testing method, explained in Chapter 4. Our proposal presents several differences with the *TimeSeriesSplit* method proposed in the Python Scikit-learn library for evaluating time-series methods, which suggests using a train set that starts at the first row and reaches a certain row r and then use from the row $r+1$ on as test. In the case of the train we suggest using sliding windows of k values to improve the efficiency. Moreover, we include the possibility of increasing this size k to accumulate in the training set a minimum number of values of each class (in our

use case a minimum amount of outliers) if needed; thus the proposed method returns splits of dynamic sizes. We also show that, after our preprocessing step, the next row to predict depends on f and is not the next one.

To the best of our knowledge this framework is not explained in detail elsewhere and constitutes the main contribution of this work.

Additionally, the third main objective in Section 1.3 involved applying the developed framework to a use case which, as explained in Chapter 2, contains stock market data. This goal was subdivided into:

1. Viability

   We assessed that, even though the use case is complex, we obtained promising results with the proposed framework:

   - For measuring the results we used a custom metric, the *gain* defined in Section 2.3.2, which discounts the growth of the stock market itself in order to determine the actual return with our method.
   - Despite not having a very high return margin, we did get a positive gain. As we saw in Figure 5.11 we reached, in the case of the Gaussian Process, a 0.3% increase per day in 3-day operations, which is definitely a promising result.
   - The *gain* score computes the revenue per day during an operation, but maybe the operations suggested by the method are scanty. In order to take this into account we suggest a second score, the *weighted gain*, that also considers the ratio of operations suggested by the method.
   - Lastly, as we saw throughout Chapter 5, the gain is directly influenced by training the model with a good set of parameters; however, finding the best parameter combination was not part of the scope of this project.

2. Monotony:

   - Backward

     In section 5.6 we discussed how, for short-term predictions, adding more past values does not necessarily improve the predictions, but rather adds extra noise and the gain decreases.

     In Figure 5.8 we could see, however, that for predictions further ahead in the future the gain takes longer to degrade. And, for longer-term term predictions (for example with $f = 20$), there is indeed a slight increase, though we expect it to also start decreasing at higher values of p.

     That being said, in order to prove this point, more experiments with higher values for p would be required, which with our current resources we could not perform. The future work section suggests some options for dealing with larger values of p.

   - Forward

     As a side effect of the experiments explained in Section 5.4 we saw how, for the three indices with which we performed those experiments, the return decreases drastically when predicting more than four days ahead, even reaching negative numbers in some cases (which corresponds to losses). This is of course expected, but should be precisely measured in a practical case for determining the best

future to predict. Although the rule of thumb obtained from our experiments might be "the closer the better", notice that taking a closer future involves performing more operations and that operations have their own fees, which are not taken into account here.

6.2. Future work

In this section we point out the following lines of future work:

6.2.1. Columns handling

During the preprocessing stage, we observed that the number or resulting columns in the generated files was proportional to the value of p. As discussed in Section 3.3, the generated file will have $p \times |Xcols|$ columns (see below a copy of Table 3.9 for column growth when increasing p).

p	# **rows**	# **cols**	**size (KB)**
1	2625	182	9.404
2	2624	363	18.743
3	2623	544	28.074
4	2622	725	37.398
5	2621	906	46.715
6	2620	1087	56.025
7	2619	1268	65.328
8	2618	1449	74.623
9	2617	1630	83.911
10	2616	1811	93.192

Even if we reduced the number of columns for the preprocessing, we would still have too many in terms of time and memory resources.

Moreover, having more columns does not necessarily guarantees better results. This is known as "the curse-of-dimensionality" (Friedman (1997)) and it occurs when we have high-dimensional spaces, where the volume of the space increases fast and the available data becomes sparse.

These issues could be handled by three methods:

1. Using dimension reduction techniques, such as Principal Component Analysis (PCA).

2. Using alternative libraries such as Apache Spark.

3. Reducing the columns considered in the training/test phase. In this work we have done this comparing the sets of columns associated to the same group of indicators (*High*, *Low*, ...) but another approach would be to examine combinations of individual columns and look for the best subset in terms of gain.

6.2.2. Outlier detection

In this project, we used mainly the model-based technique that involved calculating the mean and the standard deviation (SD) in order to find global outliers. For an α value

of 0.5, the SPX index has the outlier distribution shown in Figure 6.1. These are all global outliers, calculated over a period of 10 years.

The problem with this approach is that, for instance, a crisis period can have a very big influence both on the global mean and on the standard deviation, thus reducing the number of outliers.

In future research, it could be interesting to establish the mean and the standard deviation (or the median and the MAD) in windows of some length (a new parameter), for instance, 20 to 30 days before and after our value for detecting local outliers.

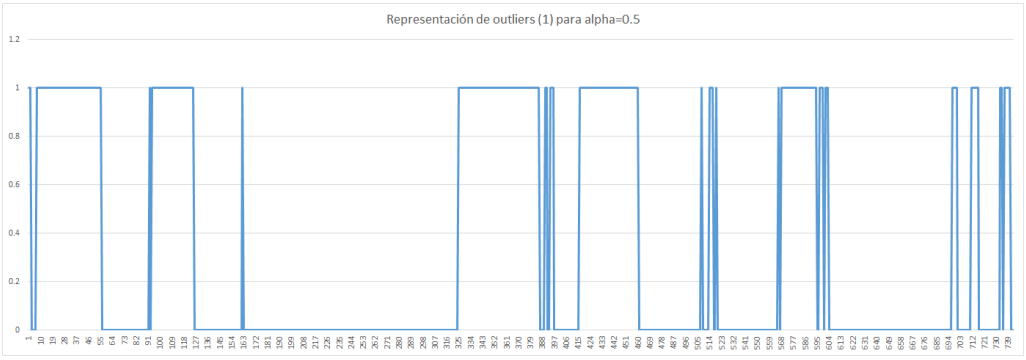


Figure 6.1: Outlier distribution for SPX with $\alpha = 0.5$

Alternatively, in Figure 6.2 we can see the daily increment of SPX for the first 200 days, along with the threshold for considering a value to be an outlier calculated with our current model. It is worth observing that, although the outlier line is very close to the mean, in fact it marks as outliers a 25% of the data. While the current model does seem to work really well, in the future other outlier detection models could be explored. One possibility is to use the *positive* standard deviation that only considers values over the mean. This makes sense if we are interested just in positive outliers, because we do not want to be influenced by the decreases in the market in the computation of the standard deviation.

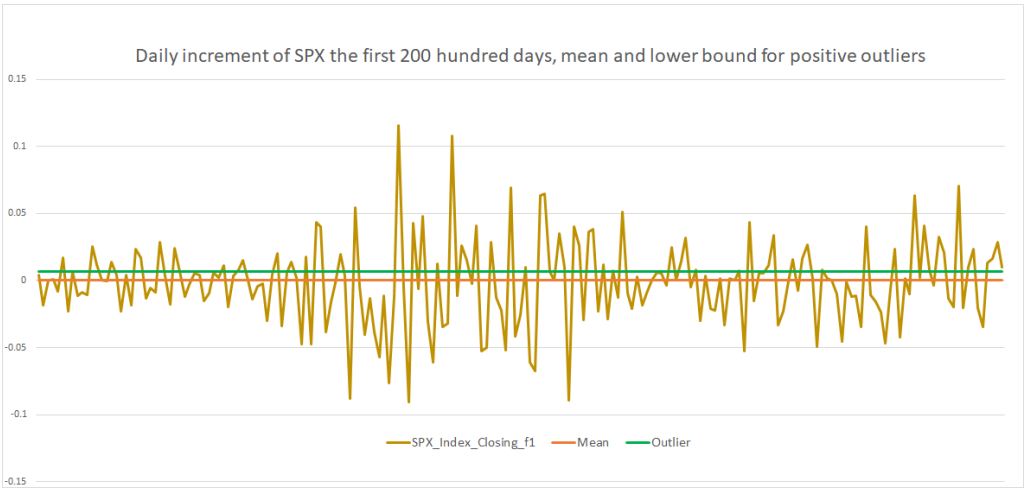


Figure 6.2: Increment of SPX with the mean (orange) and the line obtained for outliers (green) for $\alpha = 0.5$

6.2.3. Hyperparameters

In the previous chapter we analysed and discussed the results of some experiments that were carried out with different parameter combinations. However, due to the limited timeframe for this project, there was not enough time for exploring some of the paths involving hyperparameter fine-tuning. As mentioned above, the usual ML techniques for determining the value of hyperparameters are not applicable to our setting because they would involve using unknown (future) values for predicting known (past) values. Thus, an interesting line of work would be defining alternatives to cross validation for tuning hyperparameters in a time series ML framework. In the case of neural networks we could use a framework such as *Keras* (Chollet et al. (2015)) based on Tensorflow (Abadi et al. (2015)), to try different networks configurations.

In our use case, it would be interesting to do another analysis on the scaling algorithms and see if they perform better with different parameters.

Lastly, in Figure 5.11 from Section 5.8, we observed that the Gaussian Process classifier had a much better performance than the other ML algorithms, including the one we used in most experiments (Logistic Regression). This makes it a good candidate for further testing with different hyperparameter settings. In particular:

- This method admits different types of kernels (Wilson and Adams (2013)) that can have a great influence on its performance.
- The hyperparameter *warm_start* allows using part of the knowledge of the previous model when training a new one, speeding up the process.
- Another interesting characteristic of this technique is that is yields confidence intervals for the result.

6.2.4. Regression

In this project we focused on classification of detected outliers. Nevertheless, the proposed framework can be adapted for regression. The goal, in this case, would be to predict the increment of a stock market index based on its own, and other indices', past movements.

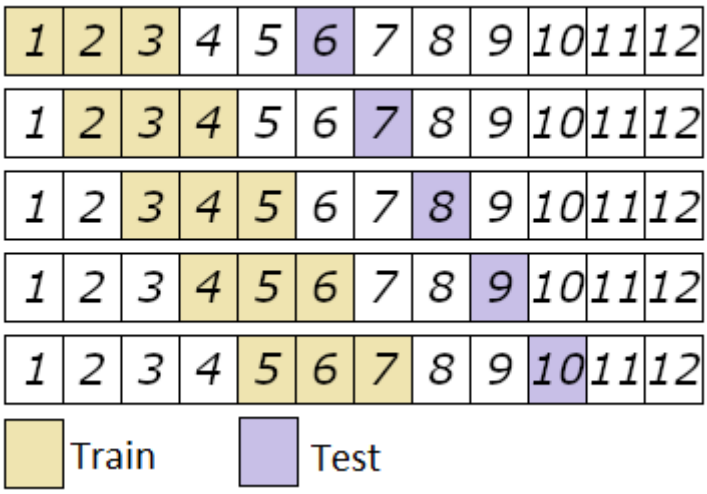
While the preprocessing phase is still valid for regression, the evaluation step would require some changes:

1. Outlier detection

   This step would no longer be necessary, as we are not looking for outliers.

2. Train/test definition

   In this case, we would use the first variant of our custom train/test splitting method explained in Section 4.2. This variant returns train/test splits on a sliding window which can be adjusted to include more or less rows for training, and maintains the required gap for the test split. Below we can see a copy of Figure 4.2, which illustrates how the sliding window works:



3. Dataset balancing

   As we want to learn from the stock market movements, balancing the dataset in this does not make sense, since we are no longer handling classes.

4. ML algorithms

   We would use or tune the ML algorithms for regression instead of classification.

5. Metrics used

   We would use the regression metrics explained in Section 2.3 instead of the classification metrics. The gain and weighted gain would still be valid metrics however.

6.2.5. Real scenario

To check our use case in practice and check if it might provide a positive revenue, it would be interesting to try it on real, up-to-date, data. A possibility would be to use the model everyday during some period, performing a simulation of real gain. Notice that in a real case scenario more factors must be taken into account, such as the fees required by the trading platforms when establishing a new operation, or the yearly tax on income.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. Software available from tensorflow.org.

Akita, R., Yoshihara, A., Matsubara, T. and Uehara, K. Deep learning for stock prediction using numerical and textual information. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, 1–6. IEEE, 2016.

Angiulli, F. and Fassetti, F. Detecting distance-based outliers in streams of data. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, 811–820. 2007.

Ayodele, T. O. Types of machine learning algorithms. *New advances in machine learning*, 19–48, 2010.

Baragona, R. and Battaglia, F. Outliers detection in multivariate time series by independent component analysis. *Neural computation*, Vol. 19(7), 1962–1984, 2007.

Basu, S. and Meckesheimer, M. Automatic outlier detection for time series: an application to sensor data. *Knowledge and Information Systems*, Vol. 11(2), 137–154, 2007.

Benkabou, S.-E., Benabdeslem, K. and Canitia, B. Unsupervised outlier detection for time series by entropy and dynamic time warping. *Knowledge and Information Systems*, Vol. 54(2), 463–486, 2018.

Bishop, C. M. *Pattern recognition and machine learning*. springer, 2006.

Blázquez-García, A., Conde, A., Mori, U. and Lozano, J. A. A review on outlier/anomaly detection in time series data. *arXiv preprint arXiv:2002.04236*, 2020.

Bollerslev, T., Engle, R. F. and Nelson, D. B. Arch models. *Handbook of econometrics*, Vol. 4, 2959–3038, 1994.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R.,

VanderPlas, J., Joly, A., Holt, B. and Varoquaux, G. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 108–122. 2013.

Carter, K. M. and Streilein, W. W. Probabilistic reasoning for streaming anomaly detection. In *2012 IEEE Statistical Signal Processing Workshop (SSP)*, 377–380. IEEE, 2012.

Chawla, N. V., Bowyer, K. W., Hall, L. O. and Kegelmeyer, W. P. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, Vol. 16, 321–357, 2002.

Chen, C. and Cook, D. J. Energy outlier detection in smart environments. In *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence*. 2011.

Cheng, H., Tan, P.-N., Potter, C. and Klooster, S. A robust graph-based algorithm for detection and characterization of anomalies in noisy multivariate time series. In *2008 IEEE International Conference on Data Mining Workshops*, 349–358. IEEE, 2008.

Chollet, F. et al. Keras. `https://keras.io`, 2015.

Friedman, J. H. On bias, variance, 0/1—loss, and the curse-of-dimensionality. *Data mining and knowledge discovery*, Vol. 1(1), 55–77, 1997.

He, H. and Garcia, E. A. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, Vol. 21(9), 1263–1284, 2009.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, Vol. 9(8), 1735–1780, 1997.

Hu, M., Feng, X., Ji, Z., Yan, K. and Zhou, S. A novel computational approach for discord search with local recurrence rates in multivariate time series. *Information Sciences*, Vol. 477, 220–233, 2019.

Hyndman, R. J. and Athanasopoulos, G. *Forecasting: principles and practice*. OTexts, 2018.

Hyndman, R. J., Wang, E. and Laptev, N. Large-scale unusual time series detection. In *2015 IEEE international conference on data mining workshop (ICDMW)*, 1616–1619. IEEE, 2015.

Ishimtsev, V., Nazarov, I., Bernstein, A. and Burnaev, E. Conformal k-nn anomaly detector for univariate data streams. *arXiv preprint arXiv:1706.03412*, 2017.

Izakian, H. and Pedrycz, W. Anomaly detection in time series data using a fuzzy c-means clustering. In *2013 Joint IFSA World Congress and NAFIPS Annual Meeting (IFSA/NAFIPS)*, 1513–1518. IEEE, 2013.

Jagadish, H., Koudas, N. and Muthukrishnan, S. Mining deviants in a time series database. In *VLDB*, Vol. 99, 7–10. 1999.

Jones, M., Nikovski, D., Imamura, M. and Hirata, T. Anomaly detection in real-valued multidimensional time series. In *International Conference on Bigdata/Socialcom/Cybersecurity. Stanford University, ASE. Citeseer*. Citeseer, 2014.

Keogh, E., Lin, J. and Fu, A. Hot sax: Efficiently finding the most unusual time series subsequence. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, 8–pp. Ieee, 2005.

Laptev, N., Amizadeh, S. and Flint, I. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1939–1947. 2015.

Li, X., Li, Z., Han, J. and Lee, J.-G. Temporal outlier detection in vehicle traffic data. In *2009 IEEE 25th International Conference on Data Engineering*, 1319–1322. IEEE, 2009.

Lin, J., Keogh, E., Fu, A. and Van Herle, H. Approximations to magic: Finding unusual medical time series. In *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)*, 329–334. IEEE, 2005.

Makridakis, S. and Hibon, M. Arma models and the box–jenkins methodology. *Journal of Forecasting*, Vol. 16(3), 147–163, 1997.

Mehrang, S., Helander, E., Pavel, M., Chieh, A. and Korhonen, I. Outlier detection in weight time series of connected scales. In *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, 1489–1496. IEEE, 2015.

Munir, M., Siddiqui, S. A., Dengel, A. and Ahmed, S. Deepant: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access*, Vol. 7, 1991–2005, 2018.

Muthukrishnan, S., Shah, R. and Vitter, J. S. Mining deviants in time series data streams. In *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.*, 41–50. IEEE, 2004.

Nelson, D. M., Pereira, A. C. and de Oliveira, R. A. Stock market's price movement prediction with lstm neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, 1419–1426. IEEE, 2017.

Ou, P. and Wang, H. Prediction of stock market index movement by ten data mining techniques. *Modern Applied Science*, Vol. 3(12), 28–42, 2009.

Papadimitriou, S., Sun, J. and Faloutsos, C. Streaming pattern discovery in multiple time-series. 2005.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, Vol. 12, 2825–2830, 2011.

Rasheed, F. and Alhajj, R. A framework for periodic outlier pattern detection in time-series sequences. *IEEE transactions on cybernetics*, Vol. 44(5), 569–582, 2013.

Reback, J., McKinney, W., jbrockmendel, den Bossche, J. V., Augspurger, T., Cloud, P., gfyoung, Sinhrks, Klein, A., Roeschke, M., Hawkins, S., Tratner, J., She, C., Ayd, W., Petersen, T., Garcia, M., Schendel, J., Hayden, A., MomIsBestFriend, Jancauskas, V., Battiston, P., Seabold, S., chris b1,

h vetinari, Hoyer, S., Overmeire, W., alimcmaster1, Dong, K., Whelan, C. and Mehyar, M. pandas-dev/pandas: Pandas 1.0.3. 2020.

Rebbapragada, U., Protopapas, P., Brodley, C. E. and Alcock, C. Finding anomalous periodic time series. *Machine learning*, Vol. 74(3), 281–313, 2009.

Reddy, A., Ordway-West, M., Lee, M., Dugan, M., Whitney, J., Kahana, R., Ford, B., Muedsam, J., Henslee, A. and Rao, M. Using gaussian mixture models to detect outliers in seasonal univariate network traffic. In *2017 IEEE Security and Privacy Workshops (SPW)*, 229–234. IEEE, 2017.

Sakurada, M. and Yairi, T. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, 4–11. 2014.

Sheppard, K., Khrapov, S., Lipták, G., mikedeltalima, Capellini, R., Hugle, esvhd, Fortin, A., JPN, Adams, A., jbrockmendel, Rabba, M., Rose, M. E., Rochette, T., RENE-CORAIL, X. and syncoding. bashtage/arch: Release 4.15. 2020.

Skryjomski, P. and Krawczyk, B. Influence of minority class instance types on smote imbalanced data oversampling. In *first international workshop on learning with imbalanced domains: theory and applications*, 7–21. 2017.

Su, Y., Zhao, Y., Niu, C., Liu, R., Sun, W. and Pei, D. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2828–2837. 2019.

Tomek, I. et al. Two modifications of cnn. 1976.

Wang, X., Lin, J., Patel, N. and Braun, M. Exact variable-length anomaly detection algorithm for univariate and multivariate time series. *Data Mining and Knowledge Discovery*, Vol. 32(6), 1806–1844, 2018.

Williams, C. K. Prediction with gaussian processes: From linear regression to linear prediction and beyond. In *Learning in graphical models*, 599–621. Springer, 1998.

Wilson, A. and Adams, R. Gaussian process kernels for pattern discovery and extrapolation. In *International conference on machine learning*, 1067–1075. 2013.

Wirth, R. and Hipp, J. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, 29–39. Springer-Verlag London, UK, 2000.

Yang, J., Wang, W. and Philip, S. Y. Mining surprising periodic patterns. *Data Mining and Knowledge Discovery*, Vol. 9(2), 189–216, 2004.

Yang, J., Wang, W. and Yu, P. S. Infominer: mining surprising periodic patterns. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 395–400. 2001.

Zhang, A., Song, S. and Wang, J. Sequential data cleaning: a statistical approach. In *Proceedings of the 2016 International Conference on Management of Data*, 909–924. 2016.